个性化你的阅读



编程狂人

Programming Madman

NO.7

推酷

关于推酷

推酷是专注于IT圈的个性化阅读社区。我们利用智能算法，从海量文章资讯 中挖掘出高质量的内容，并通过分析用户的阅读偏好，准实时推荐给你最感兴 趣的内容。我们推荐的内容包含科技、创业、设计、技术、营销等多方面内容， 满足你日常的专业阅读需要。我们针对IT人还做了个活动频道，它聚合了IT 圈最新最全的线上线下活动，使IT人能更方便地找到感兴趣的活动信息。

关于周刊

推酷周刊是专为IT人打造的行业技术周刊，目前推出的《编程狂人》是献给 广大的程序员们。 我们利用技术挖掘出那些高质量的文章，并通过人工加以筛 选整理出来。 每期的周刊一般会周一的某个时间点发布。

联系我们

tuicool2012　　164644910　　推酷网

下载APP

Android版本

立即下载 Android

iPhone版本

免费下载就在 App Store

2014/1/6 第七期

目录

业界新闻

2013 年国产开源软件 10 大年度热门项目

历史上首位程序员竟然是女的

还要骗多久？科幻大片中那些牛 X 代码真相

FileZilla Server 0.9.43 发布

Snapchat 将发布升级版应用以防止信息泄露

前端开发

一名靠谱的 JavaScript 程序员应备的素质

自己写的一个轻量级 javascript 框架的设计模式

[ISUX 转译]CSS3 transition 规范的实际使用经验

JS 闭包（Javascript Closures 译文）

js 实现 web 网页版台球游戏

编程语言

微软将推新编程语言 M#：系统编程级别的 C#

Spring 4 增强了对 Java 8、Java EE 7、REST 和 HTML5 的支持

优秀开源代码解读：RESTKit 的使用教程

Java 中的 Set, List, Map 漫谈

程序设计

Android 事件传递机制

Android 热点回顾第一期

iOS 是怎么"绘画"的?

ios 开发小技巧之提示音播放与震动

后端架构

MySQL 与 PostgreSQL：该选择哪个开源数据库？哪一个更好？

小菜白话搭架子一：有所为为 有所不为

理解事务隔离级别

数据库优化实践【高级索引、反范式篇】

优化消息生产者

程序人生

一个程序员的恐惧

如何培养创造力：别学编程，学绘画

Tinyfool 的 2013 年总结————在困惑和挣扎中试图前行

技术人攻略访谈十九：iOS 大 V 养成记

世界上最美丽的程序员+模特：Lyndsey Scott

正文

2013 年国产开源软件 10 大年度热门项目

2013 年结束了，我们根据过去一年的用户访问、交流分享和项目本身的更新频度等诸多角度对收录于开源中国的近三万款开源软件进行统计，从而得出前 10 名最受欢迎的开源软件，仅供参考。

本排行榜主要是针对国产开源软件，榜单涉及的 10 个开源软件并非同类型，尽管放在一起并不是很科学。我们只是从几个角度进行筛选，包括用户访问、软件更新和用户对该软件的讨论等。

1. Goagent

GoAgent 是一个使用 Python 和 Google Appengine SDK 编写的代理软件，goagent 的初衷是弥补 gappproxy 的不足

开发语言：Python

授权协议：GPLv2

源码下载：http://git.oschina.net/goagent/goagent

软件作者：@phuslu

上榜理由：不能说。。。

2. 禅道

禅道是第一款国产的开源项目管理软件，她的核心管理思想基于敏捷方法 scrum，内置了产品管理和项目管理，同时又根据国内研发现状补充了测试管理、计划 管理、发布管理、文档管理、事务管理等功能，在一个软件中就可以将软件研发中的需求、任务、bug、用例、计划、发布等要素有序的跟踪管理起来，完整地覆盖了项目管理的核心流程。

开发语言：PHP

授权协议：LGPL

源码下载：http://git.oschina.net/wwccss/zentaopms

软件作者：@王春生

上榜理由：不管是公司或者个人，都有非常多的用户在使用

3. JFinal

JFinal 是基于 Java 语言的极速 WEB + ORM 框架，其核心设计目标是开发迅速、代码量少、学习简单、功能强大、轻量级、易扩展、Restful。在拥有 Java 语言所有优势的同时再拥有 ruby、python、php 等动态语言的开发效率！

开发语言：Java

授权协议：Apache

源码下载：http://git.oschina.net/jfinal/jfinal

软件作者：@JFinal

上榜理由：越来越活跃，讨论越来越多，而且很多围绕该框架开发的边沿产品

4. zTree

zTree 是利用 JQuery 的核心代码，实现一套能完成大部分常用功能的 Tree 插件

兼容 IE、FireFox、Chrome 等浏览器

在一个页面内可同时生成多个 Tree 实例

支持 JSON 数据

支持一次性静态生成 和 Ajax 异步加载 两种方式

支持多种事件响应及反馈

支持 Tree 的节点移动、编辑、删除

支持任意更换皮肤 / 个性化图标（依靠 css）

支持极其灵活的 checkbox 或 radio 选择功能

简单的参数配置实现 灵活多变的功能





开发语言：JavaScript

授权协议：MIT

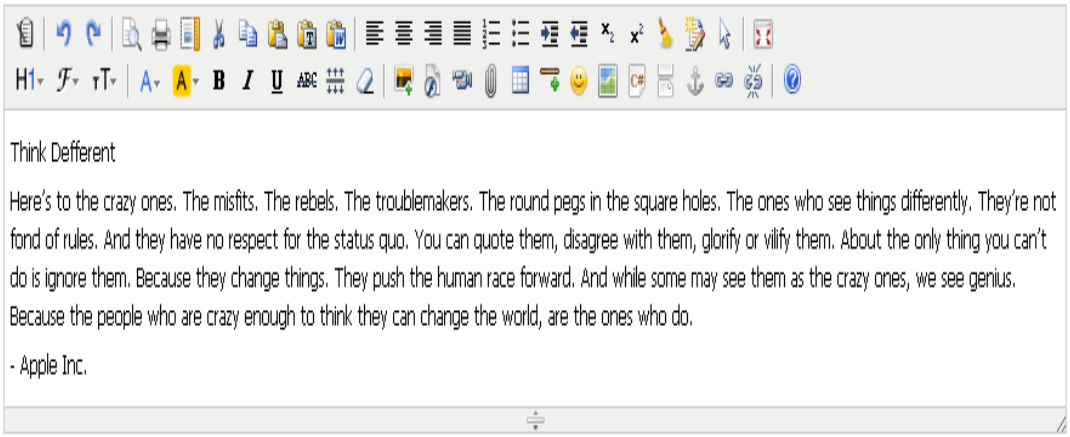
源码下载：http://git.oschina.net/zTree/zTree_v3

软件作者：@zTree

上榜理由：用户众多，更新频繁

5. KindEditor

KindEditor 是一套开源的在线 HTML 编辑器，主要用于让用户在网站上获得所见即所得编辑效果，开发人员可以用 KindEditor 把传统的多行文本输入框(textarea)替换为可视化的富文本输入框。 KindEditor 使用 JavaScript 编写，可以无缝地与 Java、.NET、PHP、ASP 等程序集成，比较适合在 CMS、商城、论坛、博客、Wiki、电子邮件等互联网应用上使用。



开发语言：JavaScript

授权协议：LGPL

源码下载：http://git.oschina.net/luolonghao/kindeditor

软件作者：@Roddy

上榜理由：用户众多，更新频繁，积极兼容新版浏览器

6. ThinkPHP

ThinkPHP 是一个快速、简单的面向对象的 轻量级 PHP 开发框架，遵循 Apache2开源协议发布，是为了敏捷 WEB 应用开发和简化企业应用开发而诞生的。ThinkPHP 从诞生以来一直秉承简洁 实用的设计原则，在保持出色的性能和至简的代码的同时，也注重易用性。并且拥有众多的原创功能和特性，在社区团队的

积极参与下，在易用性、扩展性和性能方 面不断优化和改进，众多的典型案例确保可以稳定用于商业以及门户级的开发。

开发语言：PHP

授权协议：Apache

源码下载：http://git.oschina.net/liu21st/thinkphp

软件作者：@流年

上榜理由：国产 PHP 框架中使用人数众多，很多的软件基于该框架开发

7. ECharts

ECharts 开源来自百度商业前端数据可视化团队，基于 html5 Canvas，是一个纯 Javascript 图表库，提供直观，生动，可交互，可个性化定制的数据可视化图表。创新的拖拽重计算、数据视图、值域漫游等特 性大大增强了用户体验，赋予了用户对数据进行挖掘、整合的能力。

开发语言：JavaScript

授权协议：MIT

源码下载：https://github.com/ecomfe/echarts

软件作者：@Kener-林峰

上榜理由：越来越多人开始尝试使用的图表库，讨论活跃

8. Druid

Druid 是一个 JDBC 组件，它包括三部分：

DruidDriver 代理 Driver,能够提供基于 Filter－Chain 模式的插件体系。

DruidDataSource 高效可管理的数据库连接池。

SQLParser

开发语言：Java

授权协议：Apache

源码下载：http://git.oschina.net/wenshao/druid

软件作者：@wenshao

上榜理由：来自阿里的高性能 Java 连接池，提供 SQL 监控接口，功能相当强大，使用人数众多，讨论活跃

9. Tengine

Tengine 是由淘宝网发起的 Web 服务器项目。它在 Nginx 的基础上，针对大访问量网站的需求，添加了很多高级功能和特性。Tengine 的性能和 稳定性已经在大型的网站如淘宝网，天猫商城等得到了很好的检验。它的最终目标是打造一个高效、稳定、安全、易用的 Web 平台。

开发语言：C

授权协议：BSD

源码下载：http://git.oschina.net/mirrors/tengine

软件作者：@shudu

上榜理由：来自淘宝的 Nginx 改进版，国内很多互联网网站使用

10. Pomelo

pomelo 是由网易开发的基于 node.js 开发的高性能、分布式游戏服务器框架， 也可作为高实时 web 应用框架。

Pomelo 的应用范围

pomelo 最适合的应用领域是网页游戏、社交游戏、移动游戏的服务端，开发者会发现 pomelo 可以用如此少的代码达到强 大的扩展性和伸缩性。当然还不仅仅是游戏，很多人断言未来的 web 时代是实时 web 应用的时代， 我们发现用 pomelo 开发高实时 web 应用也如此合适， 而且伸缩性比其它框架好。目前不推荐将pomelo用于大型的MMO rpg游戏开发,尤其是3d游戏，还是需要象bigworld这样的商用引擎来支撑。

开发语言：JavaScript

授权协议：MIT

源码下载：https://github.com/NetEase/pomelo

软件作者：@谢骋超

上榜理由：讨论非常活跃

原文链接

http://www.oschina.net/news/47438/2013-top-10-hot-projects-in-china?utm_source=tuicool

历史上首位程序员竟然是女的

在男性占据技术行业统治地位的今天，人们恐怕难以想象，**历史上首位电脑程序员竟然是一名女性！**



在1842年，人称“数字女王”的阿达·洛芙莱斯（Ada Lovelace）编写了历史上首款电脑程序。

据加利福尼亚大学圣迭戈分校的记载，阿达原名奥古斯塔·阿达·拜伦，是著名诗人洛德-拜伦（Lord Byron）的女儿，在她出生后不久，她的父母就离异。

她的母亲鼓励她从事数学研究，以避免像她的父亲那样出现“危险的诗人倾向”。

在1834年，阿达的朋友——英国数学家、发明家兼机械工程师查尔斯·巴贝其（Charles Babbage）——发明了一台分析机；阿达则致力于为该分析机编写算法，并于1843 年公布了世界上第一套算法。

巴贝其分析机后来被认为是最早期的计算机雏形，而阿达的算法则被认为是最早的计算机程序和软件。

她死后一百年，于1953年，阿达之前对巴贝奇的《分析机概论》所留下的笔记被重新公布，并被公认对现代计算机与软件工程造成了重大影响。

在1980年12月10日，美国国防部制作了一个新的高级计算机编程语言——Ada，以纪念阿达·洛芙莱斯。Ada 被誉为第四代计算机语言的最成功代表。

171年后的今天，电脑编程行业却被男性一统天下。女性软件开发者的收入仅为男性软件开发者的80%。

现在，只有18%的电脑科学学位授给了女性，而在1985年这个数字是37%。在所有风险投资支持的技术创业公司中，只有不到5%的公司是由女性创建的。

由于缺乏榜样的激励以及父母和老师的鼓励，**很多女性在学生时代往往会有意无意地避开电脑科学专业。**阿达女士的历史故事也许能够给她们带来一些启发。

原文 http://news.mydrivers.com/1/288/288406.htm?utm_source=tuicool

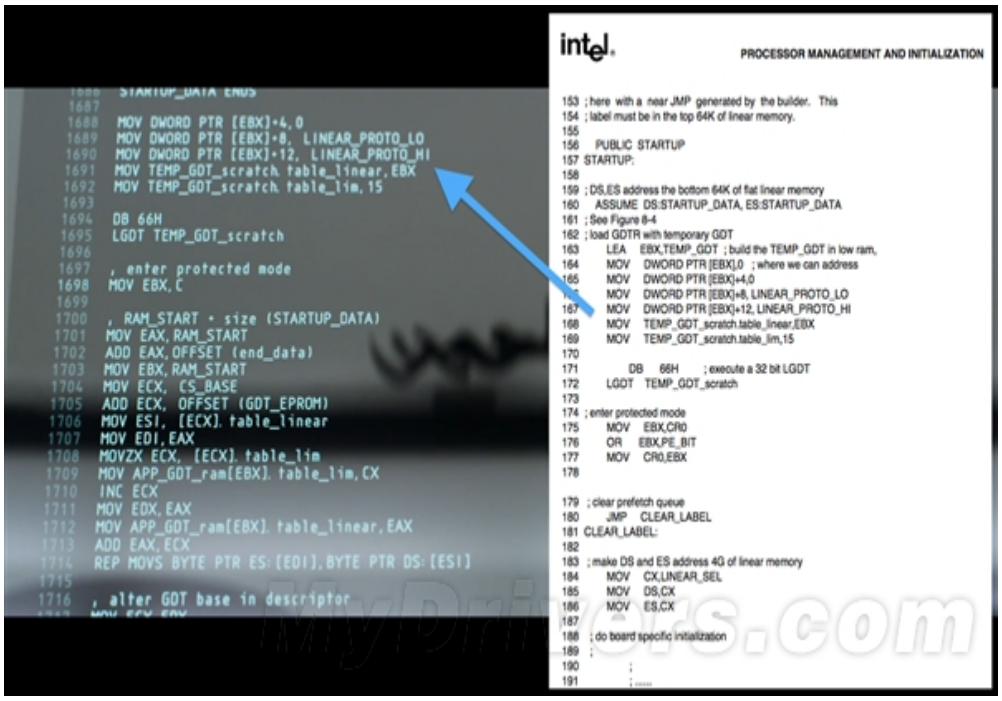
还要骗多久？科幻大片中那些牛 X 代码真相

在《黑客帝国》中，救世主 Neo 的队友通过屏幕上“1”和“0”构成的数据流，就能看到鲜活的画面，这应该算是科幻大片中对代码最极致的表现了。其他科幻电影中虽不会出现如此彻底的机器代码，但各种霸气侧漏的代码还是让观影者大赞科技的高深。

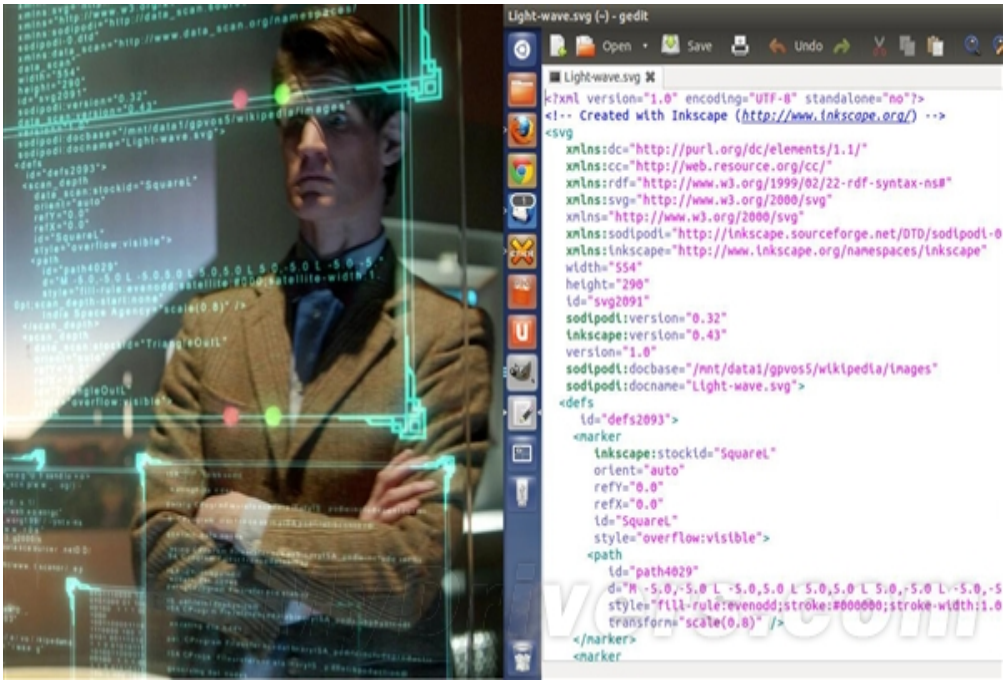
但你知道吗，**很多电影中屏幕上显示的那些牛 X 的代码实际上就是简单的从博客、维基百科上复制来的**。里面除了 C++、汇编语言、Java 语言外，甚至还有简单的 HTML 代码，但就是这些语言独有的格式排版和精简特性，让我们大开眼界。



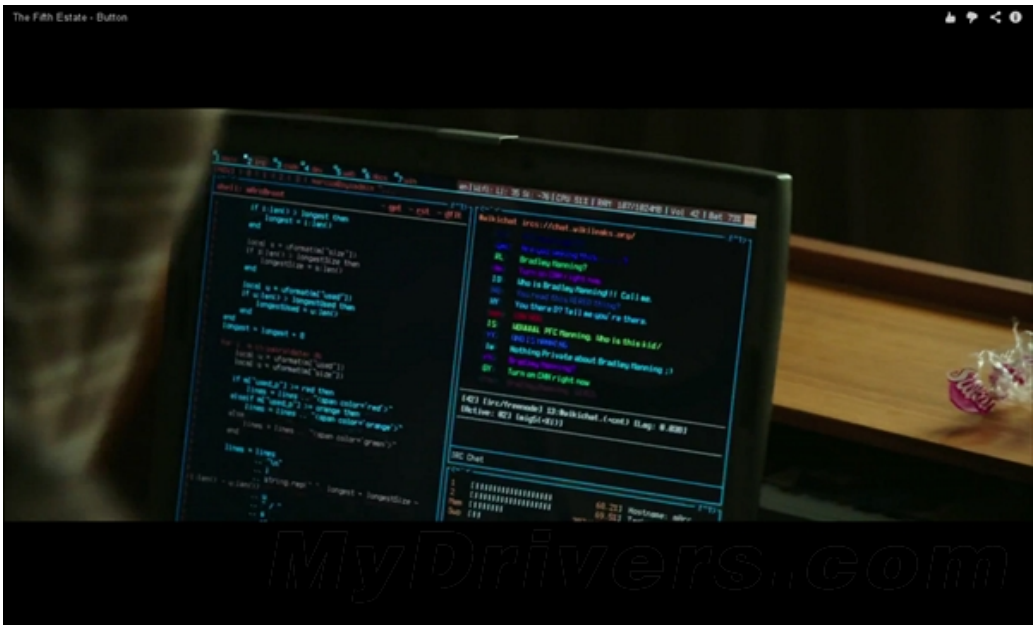
在电影《终结者》中，平视显示器中显示的代码很可能是采集于 Apple II 的一段汇编语言



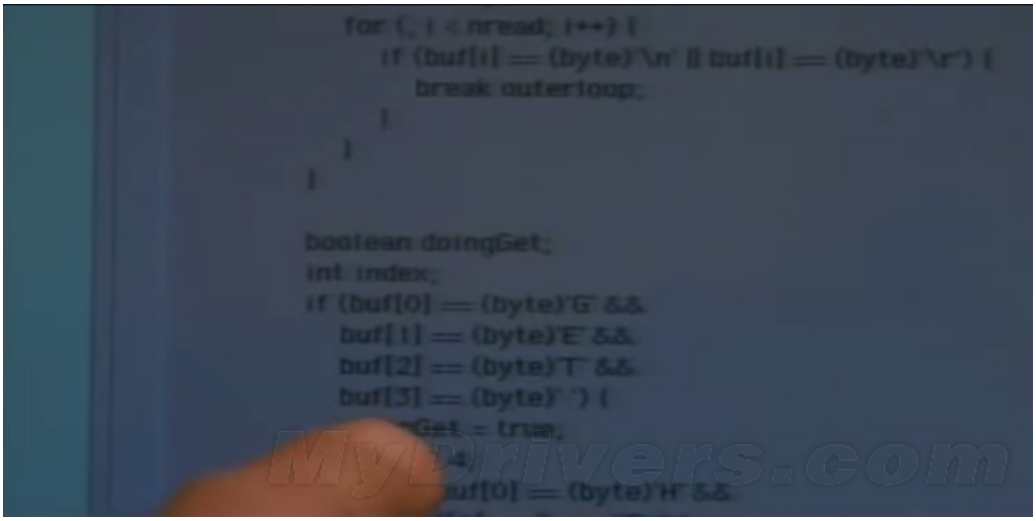
在电影《极乐空间》中，有一段系统重新启动时的代码画面，该代码实际直接摘自 Intel 系统架构开发者手册的第三卷。



在《神秘博士》的一集中，博士看的那段代码来自维基百科，是一段关于光波的描述。



电影《第五阶层》中，主人公很明显是在读一段 Lua 语言（一种脚本语言），这段代码来自 AwesomeWM 项目。



在电影《AntiTrust》（反托拉斯行动）中，屏幕上那段非常精简的代码看起来像为 WEB 服务编写的一段 Java 语句。

在美剧《Strike Back》（反击）第一季的第5集，屏幕上显示的是博客后台的一段 JavaScript 源代码。

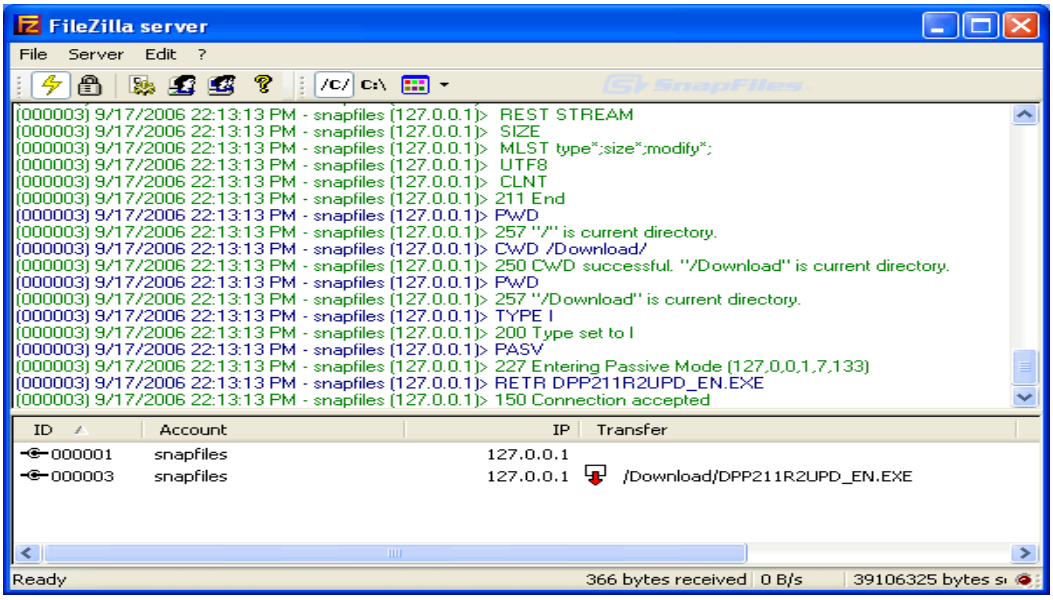
对此，我们更多的是理解，毕竟导演已经为我们展现了一个足够逼真的场景，让我们在赞叹未来高科技的同时，完成了一次身临其境的视听盛宴。更多电影代码的截图，请点击此处。

原文 http://news.mydrivers.com/1/288/288716.htm?utm_source=tuicool

FileZilla Server 0.9.43 发布

修复漏洞:

安全漏洞修复: 不允许重命名和删除了通过 FTP 命令的别名



新特性:

移除了过时的和未测试的 Kerberos GSSAPI 支持

移除了对不是 FTP 部分的非标准 OPTS UTF8 OFF 命令的支持

支持 TLS 1.2

生成认证的最小 RSA 密钥大小是 1280 位

Build system: Modernized and cleaned up workspace files for Visual Studio 2013

Build system: 移除了所有非 Unicode 的配置

Bug 修复和小改进:

Fix handling of leading/trailing whitespace in filenames

Fix display of file name at the end of a transfer

The 8+3 account setting is now stored in the correct XML element

Increase number of tries searching for a free port after the PASV/EPSV command

Fix text clipping on the miscellaneous page in the settings dialig

Fixed memory leaks when changing settings

The numbers to the PORT command are now always treated as decimal numbers as per the FTP specifications even if they have leading zeroes

FileZilla 是一款 FTP 服务器端软件，只支持 Windows 系统。

原文

http://www.oschina.net/news/47459/filezilla-server-0-9-43?utm_source=tuicool

Snapchat 将发布升级版应用以防止信息泄露

2014-1-3 12:38



昨日，Snapchat 已就最近其服务器泄露 460 万用户名和电话号码一事发表声明，将泄露原因归咎于“查找好友”服务的 API 滥用。但 Snapchat 同时也承认，自己存储信息的方式致使电话号码数据库有可能被用来与用户名进行匹配。

一家安全组织在 2013 年 8 月发布报告，指出“查找好友”功能可能存在潜在的滥用风险，随后 Snapchat 采取了限制查找频率等措施，希望能够解除这一问题。但正如 Snapchat 在此前一份声明中所指出的，攻击者能够利用‘查找好友’功能，上传大量的随机电话号码创建数据库，并与 Snapchat 用户名匹

配。

尽管攻击者在网上公布了部分删节的电话号码和用户名，Snapchat 表示：“在此次攻击中，没有任何其他的信息被获取，包括 Snaps 。”

Snapchat 并未在此次的声明中进行任何形式的道歉，该公司说：

我们将发布 Snapchat 应用的升级版，允许用户在验证手机号码后选择不在“查找好友”中显示。同时，我们改进了限制查找频率等各种限制措施，防止未来对我们服务的滥用行为。

Snapchat 社区是让朋友之间愉快地表达自我的空间。我们将竭力阻止滥用行为。

原文 http://www.ifanr.com/news/393078?utm_source=tuicool

一名靠谱的 JavaScript 程序员应备的素质

导读：很多人还瞧不上 javascript 这门简陋的脚本，文章提出应该重视 JavaScript，并提出成为一名靠谱前端工程应具备哪些能力。

关键词：javascript node.js CSS 样式

这篇文章写于2010年作者工作在 Cloudera 期间，当时 node.js 还没有流行，很多人还瞧不上 javascript 这门简陋的脚本，文章提出应该重视 JavaScript，并提出成为一名靠谱前端工程应具备哪些能力。虽然年代似乎已经有些遥远，但文中的一些观点但现在仍然适用。

Advertisement



当我尝试为 Cloudera 招一个 JavaScript 程序员时，我发现这项任务相当艰巨。找到一名真正理解 JavaScript 的人非常困难。不过，这也证明了一件非常简单的事情，是时侯去研究 JavaScript，并成为这个领域的专家了。过去的几个月，我发现了一些天才 JS 程序员并尝试去录用他们。显然他们有非常多的选择。他们被各种公司的 Offer 困扰着。作为一个想招他们进来的人，这让我想起了第一次互联网泡沫时侯的一些事。（我承诺给其中一位侯选人一辆凯迪拉克，但最终没有打动他）。最后他进入了另一家初创型公司。

比较关键的一点是：作为一名 JavaScript 专家并不能只知道 JavaScript. 如果你想专为一名靠谱的前端工程师 ，你需要在更多方面做得更好一点， 可能这些都没有写在 JD 里面，至少那些要求表现得没有 JS 那样突出。有些东西你只需要了解一下，但另外一些你需要精通(显然你需要非常精通 JS，如果你想成为一名真正的专家)。 其实，我只是一个普通人，我有博客，其他方面都跟一般人体差不多，但当我把跟 HR 说出我的要求时，HR 告诉我“这个人根本不存在，你描述的是一个超人” 我又告诉他这是我每天都在干的。 他又坚持说“你真是大神一样的人”



最后能称上 JavaScript 专家的人，终究需要一些什么技术，每天究竟需要做一些什么事，怎么判断哪些是天才，哪些是菜鸟，哪些是一般人，这是我列的一张清单：

1. 你需要知道 HTML， 你需要非常精通它。你需要知道 document types 的作用和效果。 你需要知道即使你没有写 TBODY，浏览器也会自动将他补上。你需要关注浏览器每一次升级所做的变动。学习这些东西，至少在我这儿你得这样。在你工作的时侯花点时间去研究下。

2. 你需要了解 CSS， 这样你才不至于后悔。我工作期间碰到的最多的问题就是跨浏览器的 CSS 兼容问题。一段同样的样式在不同浏览器中显示是不一样的。你添加了一个负的 margin-top 值去实现排版，猜猜怎么了？用户永远也点不到这个东西了，尽管你加了一个 z-index，尽管你可以看见这个该死的东西。或者你加了一组可以自动适应的 CSS 代码，但你要祈祷你的老板没有在 IE 里去看你的网页。最重要的一点，你要了解 CSS，这是最有价值的一项技术。你不需要非常精通，但至少能和草图做得一模一样，否则其他的事就不用提了。

3. 你需要了解浏览器，你需要明白在 IE 浏览器中，如果你把 “” 放到 DIV 中，IE 不会把它转换为 DOM 元素。但放到 header 里就可以，因为这东西是只读的。你需要明白在 Array 数组和 Object 最后加个逗号会让 IE 出错。你需要明白 IE 里 iframes 里的元素会有偏移问题。而且并不是只有 IE 会出问题，Firefox 在清除浮动时也会出问题，Chrome 里没事。你需要了解最新的 webkit 那些惊人的特性在手机和 safari 上面不一定有。

4. 我提过 JavaScript 吗?我们需要了解 JavaScript 是怎样工作的,你需要知道传入方法(function)的参数(arguments)对象是有长度的，而且是可以枚举的, 但它绝不是数组对象，更进一步，你需要知道如何将他转换为数组。你需要知道 prototype 的工作机制，并且知道在不借助框架的情况下如何使用它，并且知道它的作用和优势是什么。你需要知道垃圾回收(garbage collect)什么时侯起作用，这点很重要，还有它什么时侯不工作。你需要非常善长抽象(abstraction)。你需要非常善于为你自己和别人设计 API。你需要了解怎样测试和管理你的代码。

5. 你需要有像样的设计能力(design)：信息流， 图像设计， 用户体验设计， 代码设计。 你需要能向用户， 客户， 陌生人， 专家(随便什么人)提出正确的问题并有能力去解决他们的问题。不仅是讲出你的想法，你还要想办法去呈现给他们。你需要能操作 Illustrator 快速地把各种框架草图(wireframes)拼出一个方案来，然后你可以到 PhotShop 中(这个你需要熟练掌握)把设计图画出来(visual style)。你不需要成为世界上最好的设计师，我当然也不是。但你要有能力告诉他们什么是好的，什么是坏，什么是杰出的设计。如果你能管理好你自己好的设计，你就牛了。

6. 你需要对服务器端非常自信。其实你并不是每天都只是写写JavaScript。大多数前端工作都只需要你花一半的时间来写 JavaScript，另外一些时间你需要处理一些其它相关语言的代码. Python, Ruby, Java, PHP.. 如果你只会一种语言，那根本不值一提。如果你只会 JavaScript 和 PHP，你马上会有一些需要用到其他语言的任务(其实这是一个很好学习新事物的机会)。应该会用一种框架，但是只知道一种的话也不算真正的了解 JavaScript。

7. 然后还有许多其他的东西你需要掌握。你需要了解 Git。Git 很牛，我知道你经常听别人这样说，如果你发现一个保守的人拒绝去学新的知识扩展自己，你需要让他明白他的错误。你需要知道一些 shell 脚本。在没有记事本，TextMate 或其他编辑器的情况下，你需要能够打开 VI 或者 Emacs 去修改一个配置文件(conf)。你需要知道怎么去编绎一些东西，怎么样去截取一个日志文件，解压一个压缩包(tarball)，加密一些东西，开启并配置服务。你需要有能力去关掉(kill)进程并跟踪调试一个堆栈。

我甚至算不上一个熟悉敏捷开发流程，参与过开源软件项目或发布新版本的程序员，不过如果你认为你能够匹配我所列出的7条中的5条，或者全部匹配。那么，恭喜你，你就是传说中的大神。

那么你成为大神后你会得到什么?你怎么知道多年处理那些乏味的 IE6的 BUG 是值得的?好吧，你得到的是，你会每周都收到 HR 的邮件和电话。(这其实并不稀奇，他们就像懂你的销售)。如果你想换工作的时侯，你甚至什么都不用做，告诉一小搓朋友或者发到你的 twitter 上，然后很多扇门就在你面前打开了。你会有些新朋友，他们希望你加入他们的事业。当你加入时你会成为传说中的元老。你会得到股票，并设计一款产品的最初版本。你会得到一辆1967年的凯迪拉克作为签约的报酬。在公司刚起步时我曾经诱惑过这样的家伙。大神是一名还不存在的产品的前端工程师。他指出一两年后这东西值很多美元，然后多年以后，他可以告诉别人，“这是我做的”。现在已经有这样的一些人了，像已经有无数人使用的 Twitter, Facebook, Gamil 和 Google Maps，像 Iphone 的界面，Github, YouTube，肯定有人说，“这是我做的”。

原文链接

http://www.searchsoa.com.cn/showcontent_79478.htm?utm_source=tuicool

自己写的一个轻量级 javascript 框架的设计模式

公司一直使用 jQuery 框架，一些小的项目还是觉得 jQuery 框架太过于强大了，于是自己周末有空琢磨着写个自己的框架。谈到 js 的设计模式，不得不说说 js 的类继承机制，javascript 不同于 PHP 可以轻松的实现类继承，不过 javascript 的类继承方法还是有的，常见的有构建函数、原型扩展、综合……，也有一些专门写类的函数，例如 jQuery 的作者有个类继承函数。关于类继承写法可以简单看看这篇文章。

我自己写的框架也不知道叫什么名字，刚开始写的时候随手写了个 W（姓拼音，之前写过 jQuery 弹出框插件 wBox），现在写这篇文章重新整理了一下思想，换了 YQ（名字拼音，你懂的~）。



核心代码

框架设计的时候尽量做到了支持链式写法，也就是返回 this，可以 $(selector).handler1().handler2()……无限写下去，只要不是有返回值的就可以继续。包括了 event，dom，css，还有 fadeIn，fadeOut 动画（因为再重构爱墙[html5+css3]版，所有顺手加上了这个功能）。如果配合 sizzle 选择器就更牛了！

下面说说框架的核心代码，等完善了之后跟新版 html5爱墙一起放出，主要的代码如下：

```
(function(window,document){
    var DOC = document,YQ = window.$ = function(selector){
        if(!selector){
            return YQ.elems
```

```
            }
            typeof selector === 'string' && (selector =
getElements(selector));//简单的判断是 dom 对象，还是字符串，字符串则开
始选择器
            return superElems(selector);
        }
        function superElems(elems){
            if(elems){
                if(elems.nodeType){//判断是否为 DOM
                    if(typeof elems.getAttribute !=="unknown"){
                        var temp = elems;
                        elems = [];
                        elems[0] = temp;//把本身作为超级对象的第一个，
其他方法进行扩展
                        for(var i in YQ.elems){
                            //扩展对象，保留已有的方法
                            typeof elems[i] === "undefined" && (elems[i]
= YQ.elems[i])
                        }
                    }
                }else{//如果为对象，则对对象进行扩展
                    elems = YQ.extend(elems,YQ.elems);
                }
            }
            return elems;
        }
        function getElements(selector){
            //伟大的选择器，可以使用 sizzle
            var dom = DOC.getElementById(selector);//……
```

```
        return dom;
    }
    YQ.tool = {
        isFunction:function(obj){//简单的判断是否为函数
            return obj && typeof obj ==="function";
        }
    }
    //此处为超级对象一些扩展
    YQ.elems = {
        each:function(dom,callback){//强大的 each
            if(YQ.tool.isFunction(dom)){
                arguments.callee.call(this,this,dom);
            }else{
                for (var i = 0, len = dom.length; i < len; i++) {
                    callback.call(dom, i, dom[i]);
                }
            }
            return this;
        },
        find:function(selector){
            var doms = [];
            this.each(function(i,dom){
                doms.push(YQ.DOM.find(selector,dom));
            })
            return superElems(doms);
        }
    }
    YQ.each = window.Array.prototype.each = YQ.elems.each;//扩展
Array
```

```
YQ.extend = function(subClass,baseClass){
    for(var i in baseClass){
        subClass[i] = baseClass[i];
    }
    return subClass;
}
YQ.AJAX = {}
YQ.CSS = {
    names:{
        'float':'cssFloat',//区分cssFloat or styleFloat
        opacity:'opacity'
        //.....
    }
}
YQ.browser = {
    isIE:'',
    isFirefox:'',
    version:'3.6'
    //.....
}
YQ.event = {
    names:{
        mousewheel:YQ.browser?"DOMMouseScroll":"mousewheel"
    },
    fix:function(e){
        if(e && e.clone) return e;//如果已经处理了，直接返回
        e = window.event || e;//event 是全局变量
        var fixE = {
            clone:true,
```

```
                stop:function(){//冒泡
                    if(e&&e.stopPropagation){
                        e.stopPropagation();
                    }else{
                        e.cancleBubble = true
                    }
                },
                prevent:function(){//默认动作
                    if(e && e.preventDefault){
                        e.preventDefault();
                    }else{
                        e.returnValue = false;
                    }
                },
                target:e.target || e.srcElement,
                x:e.clientX || e.pageX,
                Y:e.clientY || e.pageY,
                //鼠标滚轮事件统一处理
                wheel:e.wheelDelta/120 || -e.detail/3
            }
            return fixE;
        }
    }
    YQ.DOM = {
        find:function(selector,parentDom){
            //do something
        }
    }
})(window,document);
```



简单的分析

代码中的注释还算详细，结合注释就可以看懂，下面我再多说几句，高手飘过，欢迎拍砖……

总得来看是个匿名函数，定义了全局变量$（似乎都喜欢美元，如果有喜欢的人民币的，下次加上），函数里面有个 YQ 的对象，有很多 function，包括一些私有的。

使用$可以作为选择器，如果为空，则返回 YQ.elems 对象，选择后的 DOM 根据 YQ.elems 进行扩展，最后得到的就是注释里面说的 superElems（此处创建 superElems 方法参考了下 jRaiser），跟 jQuery 的 superElems 类似，绑定了很多方法，可以方便的进行操作，YQ 的方法可以通过 YQ.elems 扩展到选择器选择的对象上去，而选择器主要的函数 getElements，可以使用 sizzle，这样就方便了 dom 操作，不过 sizzle 似乎也是代码太多，此处将来会支持简单的标签,#ID,.class,标签+class 选择器。

另外框架中中对一些 Array，string 进行了扩展，例如上面的 Array.each 方法，YQ.elems.each 是重量级的函数方法，可以支持 superElems 的遍历（返回本身），还可以支持简单数组的遍历。

框架中还对一些兼容性进行了处理，例如 YQ.event.fix 处理事件，css 中处理 float

基本就这些了，今天就先说核心的部分，其实通过这段代码已经很清晰的了解到这个框架的思想了，后续继续给力的 coding，补充扩展……

目前为止,框架不到800行代码,包括了常见的 jQuery 方法,压缩后不到9k,gzip 4.5K~

原文 http://blogread.cn/it/article/3884?f=hot1&utm_source=tuicool

[ISUX 转译]CSS3 transition 规范的实际使用经验

本篇文章主要讲述 CSS3 transition 规范和在不同浏览器之间的使用差异，关于具体解决方法或如何规避问题的意见可以参考另一篇非常有见地的文章，“ All You Need to Know About CSS Transitions ”。Alex MacCaw 讲述的是关于实现特定的效果，而我要谈的是技术背景，主要讨论在使用 CSS 过渡的过程中所未预料到的问题。

结构 (HTML)，表现(CSS)，以及行为(JavaScript)相分离并不是什么新鲜的事情，然而 CSS 能跨越这个界限并且可以在短期内得到实际的应用，这还真的是一个完全不同的讨论话题。

几周前，我开发一个 JavaScript 模块，在能够使用 CSS 过渡的条件下，JavaScript 端又无法获取到实现过渡的方式。实际遇到的问题是这两者根本没有办法同步，经过多次的测试后，我只能放弃。而我的测试结果正是本文所讲述的。

首先，我们要说一下 getcomputedstyle()，是一种用 JavaScript 返回浏览器渲染 CSS 的属性值的方法。 这个方法可以查看“ DOM Level 2: getComputedStyle() ”和“ CSS Level 2: Computed Values ”。

这对于像 font-size 这样的属性，通过一个参数便可以转换为像素值。但对于可以缩写的属性值，例如 margin ，一些浏览器则返回为空。再就是那些同一属性的不同属性值，例如 font-weight 的值 bold 和700。WebKit 也有一个小 bug，它会从伪对象中提取出属性值。

这里所讲述的浏览器之间的差异是2013年1月在使用 Firefox18 (Gecko)，Opera 12.12 (Presto)，Internet Explorer10 (Trident)，Safari 浏览器6.0.2 (WebKit)，Chrome 23 (WebKit) 以及 Gecko 和 WebKit 的 Nightly build channels。

事不宜迟，让我们来一起看一下规范与实际情况的差异，为了方便，我省略了各浏览器的前缀。在文中我通过创建一个 CSS3 Transitions Test Suite 来发现问题。

1、指定过渡

CSS3 transitions 规范定义了以下四个 CSS 属性：

- transition-property
- transition-duration
- transition-delay
- transition-timing-function

过渡属性

transition-property 是用来指定当元素其中一个属性改变时执行 transition 效果。系统默认值是 all，这意味着浏览器能够以动画形式呈现所有的可过渡属性（transition-duration 持续时间超过0s），该属性支持单个值或以逗号隔开的多个值列表（跟其他所有 transition-*属性一样）。

规范规定，一个浏览器应该接受并保存任何它不能识别的属性。因此，下面的例子中将会看到持续2秒的 padding 过渡：

```
    transition-property:foobar,paddin
1
 g;
2
    transition-duration:1s,2s;
```

不同于规范的是，上面的情况在 WebKit 下会解析为 transition-property: all。 而 Firefox 和 Opera 会解析为 transition-property: all, padding.

过渡持续时间

transition-duration 属性规定了一个过渡从初始状态到目标状态的持续时间。它接受以秒或毫秒的<time>值（例如，2.3S 和2300ms 都是指2.3秒）。

尽管规范明确规定了过渡值必须为正数，但 Opera 仍接受-5S 的值，至少对于 getComputedStyle（）来说是这样的。虽然规范中并没有限制属性值的大小，但 Opera 和 IE 不接受低于10ms 的值。而 WebKit 在 getComputedStyle（）执行中有个小 bug，例如：返回值0.009999999776482582s 会取代0.01s。

过渡延迟时间

transition-delay 属性规定了在执行一个过渡之前的等待时间，同样使用<time>值。Delay 可以是负值，但这会导致动画无法平滑过渡。

IE 和 Opera 不接受 transition-duration 在-10ms 和10ms 之间的值。WebKit 的 floating point 也会在这儿出现。

过渡时间曲线

transition-timing-function 属性规定了过渡效果的时间曲线。包括 cubic-bezier（x1, y1, x2, y2），step（<number>, start|end），和预先定义的 cubic-bezier 曲线关键词，linear, ease, ease-in, ease-out 和 ease-in-out。在使用 LEA Verou 特有的 cubic-bezier 曲线编辑器时，cubic-bezier 背后的公式就变得不再重要。尽管 cubic-bezier 曲线会平滑过渡，但是 step（）函数会在一个固定的间隔跳到下一个值。这样便会产生逐帧动画的效果；如“ Pure CSS3 Typing Animation With steps() ”。

linear 的计算值通常表示为 cubic-bezier（0, 0, 1, 1）—— WebKit 除外。但 WebKit 仍然会返回 cubic-bezier（0.25, 0.1, 0.25, 1），而不是 ease。规范规定 X 值的必须介于0和1之间，y 值可以超过该范围，而 WebKit 允许 X 超过此范围，而 Android 浏览器（4.0版本）却混淆了 x 和 y 的范围。

2 过渡完成

我前面已经提到了 CSS 过渡异步运行的问题。规范提及了 TransitionEnd 事件允许 JavaScript 与已完成的过渡同步进行。但可恶的是该规范对此并没具体阐述。事实上，它只是简单地说明单个事件会因为已完成过渡的属性而被终止。

规范指出缩写属性（如 padding）应为包括其在内的所有属性（padding-top, padding-right，等等）实现过渡，它并没有说哪个属性应该在 TransitionEnd 事件中被具体命名。然而即使过渡被定义为缩写属性（如 padding），Gecko，Trident 和 Presto 对于普通书写的子属性（如 padding-top）同样可以实现过渡，而 WebKit 则会阻止过渡。 如果你指定 transition-property: padding，WebKit 会为 padding 执行过渡，但 transition-property: all 这样就会针对 padding-left 执行新的过渡。而当 padding 正执行过渡时， iPhone 6.0.1 的 Safari 浏览器在也可以执行 font-size 和 line-height 的过渡。

```
    .example{padding:1px;transition-property:padding;transition-duration:1s;}
    .example:hover{padding:10px;}
```

以上 CSS 将在不同浏览器下触发不同的 `TransitionEnd`：

Gecko, Trident, Presto:

padding-top, padding-right, padding-bottom, padding-left

WebKit:

```
1 .example { padding: 1px;transition-property: all, padding;transition-duration:1s;}
2 .example:hover{padding:10px;}
```

以上 CSS 将在不同浏览器下触发不同的 TransitionEnd：

Gecko, Trident, Presto, WebKit:

padding-top, padding-right, padding-bottom, padding-left

Safari 6.0.1 on iPhone：

padding-top，padding-right，padding-bottom，padding-left，font-size，line-height

你可以指定负值 transition-delay 来“快速实现”转换。但是 transition-duration: 1s; transition-delay: -1s; 在 Gecko 和 WebKit 下执行转换并会立即跳转至目标值。而 Trident 和 Presto 将不会触发任何事件。

WebKit 在 getComputedStyle() 上遇到的浮点问题也同样存在于 TransitionEnd.elapsedTime 中，所有的浏览器如此。Math.round(event.elapsedTime * 1000) / 1000 可辅助修复。

WebKit 和 IE 浏览器下执行 background-position，会触发对 background-position-x 和 background-position-y 的 TransitionEnd，而不是 background-position 的 TransitionEnd。

所以，即使你知道过渡正在执行，你也不能依赖已有的 TransitionEnd.propertyName。尽管你可以编写大量的 JavaScript 来弥补，但在没有对每一个属性进行恰当性能检测的情况下，即使你采用最新方法也将无法实现。

3 过渡属性

规范列出了浏览器支持动画过渡的一些 CSS 属性。当然也包括 CSS2.1的属性。还有一些可以动态变化的新属性，如 Flexible Box Layout 。

该属性数值类型非常重要。margin-top 接受<length>和<percentage>值，但根据可过渡 CSS 属性列表，只有<length>是可实现动画效果。但这并不能让浏览器开发商避开<percentage>值实现过渡。然而，word-spacing 属性除外。该

属性包括<percentage>值，但没有浏览器能以动画形式显示。

撇开 TransitionEnd 事件，如果在过渡发生的指定时间内，getComputedStyle（）值从 A 变到 B，该属性就会从值 A 过渡为值 B。如果没有执行，例如“CSS 属性值发生变化”，那么也许应该仔细核查下 DOM。setTimeout（）的解析度还不够好以达到快速过渡（小于几百毫秒的持续时间），这时候 requestAnimationFrame() 就是你的帮手。在重绘前会提醒你，并提供了一些中间值供参考。除了 opera，其他的都可以支持。

4 过渡属性的优先级

transition-property 规范允许多次过渡单个属性，如果单个属性在“过渡属性”中的值被多次指定，过渡将通过持续时间，延迟和时间曲线给出的值来实现。因此，我们可以实现 padding 过渡持续1秒，padding-left 过渡持续2秒；或使用 transition-property: all 来定义默认属性类型并重置特定属性。

在 Firefox 和 IE 浏览器上，这些都没有任何问题。 但 opera 下会混淆优先顺序。它认为 padding-left 比 padding 和 all 更加具体，而不是简单地使用最后一个属性。

最大的问题是 WebKit。如果一个属性被多次指定，它将进行多次过渡。 如果想让 WebKit 崩溃，尝试用 transition-duration ： 0.1 秒运行 transition-property: padding, padding-left，WebKit 将至少执行两次过渡。但更有意思的是，TransitionEnd 可以进行上百次的单一过渡。

5 auto 的转变

CSS 属性中的 auto 值能够自适应宽度，如果块级元素设置了 width: auto，那么就会继承父级的宽度。有时你需要从 width: auto 改变到一个具体宽度，并且需要过渡那个改变。当然本规范并没有强制或否定 auto 值可用于过渡。

Firefox，IE 和 Opera 无法从 or 值过渡到 auto 值。 IE 下有 z-index 有一点点例外，但仅此而已。 另一方面，WebKit 能够从 and 过渡到几乎可以接受 auto 值的任意 CSS 属性。WebKit 不太喜欢 clip；因为这个属性，它只会引发 TransitionEnd 过渡，而过渡期间不会产生或显示任何中间值或状态。

对于其他属性，如 width 和 height，WebKit 下会有一些差异。如果 width: auto 过渡为 300px 的宽度，然后再过渡成 100px，那么过渡不会从 300 缩至

100 像素。它会从 0 增加到 100 像素。

关于完整的兼容性列表，可以查看“ CSS Animatable Properties .”，

6 隐式过渡

隐式过渡发生在当一个属性的改变引起另一个属性被过渡的时候，或者当你想改变一个父级元素中的属性，会导致子元素不论是继承过渡或附属属性的过渡。font-size: 18px, padding: 2em——padding 会被计算为 2 × font-size， em 就是36像素。

有各种各样的相对值类型：<percentage>, <length>, em, rem, vh, vw 等等。使用一个相对值，如 padding: 2em，让浏览器重新计算属性的 getComputedValue()，每次应变量（如 font-size）都会发生改变。由于计算样式改变,将反过来导致 padding 的过渡。这种过渡被定义为“隐式过渡”，因为 padding 属性值没有被修改。

大多数浏览器会实现这种隐式过渡。除了 IE 10，只对 line-height 属性执行隐式过渡。除了 vertical-align 外，Webkit 可以针对其他所有属性执行隐式过渡。除了字体相对属性值，还有宽度相对属性值（通常为<percentage>），相对属性值（如 vh 和 vw），默认初始值（Opera 中的 column-gap: 1em），还有“ currentColor ”。所有这些都有可能会，也可能不会引起隐式过渡。

在 Firefox 中，当继承和附属属性执行过渡，但他们的 transition-duration 或 transition-delay 并没有随着过渡，这些隐式过渡就会变得特别有趣。 而 Webkit 和 Opera 执行过渡时会很有视觉感，但 Firefox 会稍显错乱。在 IE 中不会轻易执行隐式过渡。

另外，别忘了继承，DOM 元素的 font-size 将会由其子元素继承，只要不被覆盖，就可能引起隐式过渡。

7 转换和伪元素

伪元素（:before 和:after），在 CSS2 中已经有了介绍. 如果不熟悉可以查看 “ Learning to Use the :before and :after Pseudo-Elements in CSS ”。虽然 CSS3 中定义了额外的伪元素（::alternate, ::outside），但是他们（到目前为止）还并没有被支持。因此所有 CSS 动画属性也应该是伪元素的动画属性。

Firefox 和 IE 10 可以在伪元素上实现属性过渡. 而 Opera，Chrome 和

Safari 则不会。 WebKit 从2013年一月起也开始支持。

伪元素的过渡会导致内容自身产生一些新问题，因为在生成内容时 TransitionEnd 过渡根本还没有结束。 在某一时间段内，他们理应在主元素上被触发，并通过 TransitionEnd.pseudoElement 提供伪元素，但即便是“CSS 动画过渡”的“过渡事件”部分，编写者的方案也并没有指定哪一个最合适。

我们想要改变 content 属性值，因此 IE 8将在特殊情况下（比如:hover 状态）将会重新渲染该元素。结果表明，对老的 IE 版本进行兼容会影响到所有其他浏览器的效率。所以，当试图在伪元素上进行属性过渡时，要确保 content 的值不会被改变。

如果主元素没有运行:hover 状态，那么 IE 10 将不针对伪元素“:hover”执行过渡。

```
1 .some-selector:before{content:"hello";color:red;transition:all 1s linear 0s;}
2 .some-selector:hover:before{color:green;}
```

在 IE10 下，:before 在 mouseover 的时候，:hover 是一定要定义的。

这个问题在于不是一定要求你定义主元素:hover 状态。而是如果没有定义，IE 10 会将:hover 解释为:active。更奇怪的是，:active 状态甚至会在 mouseup 后继续持续，而当你再次点击就会取消。

8 背景标签

在编辑标签时，IE 10 是唯一可对背景或前景响应的浏览器，如果标签变为背景后，虽然它会完成正在执行的过渡，但它不会执行新的过渡。IE 10 将等到标签变为前景后再执行新过渡。幸运的是，IE 10 已经支持页面的可见性 API，允许开发人员应对这种操作行为。

9 隐藏元素

对于隐藏的元素，过渡是不会被执行的，因为大多数浏览器都明确认为没有必要在一个看不见的元素里运行过渡。然而，也有特例，在 Opera 下无论元素隐藏与否它都将执行过渡。

10 过渡之前，DOM 树是否加载完毕

当文档脱离解析模式时，DOMContentLoaded 被触发，如果你在使用 jquery，

那么应该了解 jQuery.ready() ，过渡可以在这之前运行。

11 渲染差异

这个问题我之前已经说过了，本文就是基于我的测试结果进行阐述的。测试是自动运行的，但事实证明，还是发现了很多问题。

当时要实现从渐变到渐变的背景过渡是不可能的，但可以实现从渐变到纯色的过渡。如果渐变正在进行中，从白色到目标颜色的过渡即将开始，在过渡启动时，会看到白色在快速闪动。目前所有的浏览器中都可以察觉到这一点。

不过 Firefox 似乎是用不同的算法来渲染图像的，以表明它们执行了动画过渡（ 见实例）。很显然，在动画过渡时， Gecko 并没有呈现好的效果。如果 transform: scale() 足够低，这种情况将发生。

Firefox 不会从 a:visited 到 a:hover 过程中过渡动画，反之亦然。 但它会从 a:visited 直接跳到 a:link，然后过渡到 a:hover 状态， 你可以在这个 例子 中看到，这是在 Mozilla Developer Network” Privacy and the :visited Selector ”中提到的。然而 IE 10 与 Chrome，Safari 和 Opera 浏览器一样，会从 a:link 到 a:visited 实现过渡。

如果子元素的 position 改变时， Firefox 不会触发元素的属性，而 WebKit，Opera 和 IE 10 则会触发。

12 对规范的建议

看完了整个规范并对所有功能进行了测试之后，觉得如果能进行以下优化将会更好：

- 加入 TransitionsEnd（注意是复数），一个元素的所有过渡一旦完成就启动触发。它能告知一系列已被触发的属性，但是不需要知道哪些已被过渡， 只要知道所有的动画过渡何时可以完成即可。
- 加入 TransitionStart 任务，以便可以获取每个待过渡属性。因为 JavaScript 的事件循环和渲染路径不一定能互相牵制， 单一的 TransitionsStart（也会重复多次）可能是更好的解决方案。我不知道为什么要 cancel 任务，所以这就叫“操作后就不再管”。
- 要明确哪些 TransitionEnd 需要被触发，前面举例的 WebKit 中 padding 和 padding-left 的问题会让人很头疼。

- 要明确说明“隐形过渡”如何处理，前面例子中 transition-property: font-size 的 line-height: 1em 应该要有明确的处理方式。
- 需要添加那些允许定义 pointer-events: none 并防止意外悬停状态的::transitioning 伪类，这里防止滥用样式，因为他们自身会引发新的过渡或者改变正在进行的过渡。除了这些建议，我们还需要能在不大量使用 JavaScript 进行辅助的情况下进行一些常规操作。
- 有时候你需要禁用过渡。例如，为了在网站访问者面前呈现完美过渡之前，你需要调整布局并对尺寸规格进行精确测算对位置进行完美布局。
- 有时你想立即从 DOM 中移除一个对象。你可以添加一个类，等待 TransitionEnd 完成后再进行删除。
- 跟删除对象一样，你想要添加一个新元素。你可插入这个元素，设置“隐藏”以实现新元素的样式变化。
- 重新排序，隐藏和显示元素都比较常见。针对这些进行样式操作就要像操作实用程序一样，如 Isotope 。

13 使用 delay

使用延时，可以很好的解决无意的鼠标悬停造成的样式变化，如同 setTimeout()。

14 总结(可参考前面谈到过的实例)

- 使用 transition-property: all 时注意，否则将遇到本不需要进行转换的 TransitionEnd 情况。
- 当使用可缩写属性时，触发事件的数量会根据不同浏览器而不同。
- Opera 和 IE 不支持延迟时间为负值。
- Webkit 在属性优先级上存在问题，例如：要避免 transition-property: margin, margin-left 的情况。
- IE 不支持隐式转换。
- Firefox 和 Opera 无法解析 transition-property: all, width。
- Opera 混淆了属性的优先级。

原文 http://www.tuicool.com/articles/fIJNfm?utm_source=tuicool

JS 闭包（Javascript Closures 译文）

Javascript 闭包

简介

“闭包”就是一个表达式（通常是一个函数），该表达式可以自由访问一些变量和变量所处的环境（这个环境将这些变量‘close’在了里面）。

闭包是 JS 最强大的特性之一，但在没有深刻理解它之前我们很难充分、高效地利用它。闭包通常很容易被创建，有时甚至是无意识地，但这种创建的后果通常是有害的，尤其是在 web 浏览器环境中。为了避免在无意识的情况下引发闭包的缺点并充分利用它的优点，我们有必要了解闭包底层的机制。该机制主要是作用域链(scope chain)在解析标识符（identifier）及对象属性时的工作流程（This depends heavily on the role of scope chains in identifier resolution and so on the resolution of property names on objects.）。

关于闭包最简单的解释就是 ECMAScript(Javascript)允许内部函数（inner function），内部函数是指函数声明或函数表达式（关于函数声明或函数表达式的区别大家可以先百度一下，改天有空的话我再单独写篇文章）是在另外一个函数（称为 outer function 或外部函数）的函数体中定义的。该内部函数可以自由访问该 outer function 的函数参数、局部变量 及 其它内部函数。当这些内部函数在它的外部函数之外被访问时，闭包就形成了。这意味着这些内部函数在它们的外部函数返回之后依然可以被执行。当这些内部函数被执行时，它们所能访问的外部函数的参数、局部变量及其它 inner function 的值都处于外部函数返回之前的状态（相当于是保留了一份当时的现场）。

对象的命名属性解析

ECMAScript 共有两种类型的对象，分别是**原生对象（Native Object）**和**宿主对象（Host Object）**，其中原生对象是指 js 语言本身的对象（Array, Date 等内置对象，以及通过 js 代码创建的对象如 var obj = new Object();等），宿主对象是指 JS 执行环境（客户端环境通常是浏览器，服务器端可能是 Node 等）所内置的对象，例如浏览器环境中的 window 对象，dom 对象等。

原生对象是松散动态的**命名属性**的集合（Native objects are loose and dynamic bags of named properties），这些命名属性的值可能是其它对象的引

用，也可能是一个原始值：string，numbe,boolean，null 或 undefined. 其中 undefined 这个原始值有一点怪异，当我们把它赋值给对象的某个属时,并不会将该属性从对象中移除，而只是该属性的值是 undefined 而已。

下面的这个例子展示了如何给对象的属性赋值以及如何从对象中读取这个属性的值。

赋值

```
//创建一个普通的 js 对象
var objectRef = new Object();
//给该对象添加一个名为'testNumber'的属性：
objectRef.testNumber = 5;
/* - 或:- */objectRef["testNumber"] = 5;
```

//在上面这句赋值语句执行之前，objectRef 所引用的对象上并没有 *testNumber* 这个属性，但在赋值之后它有了。之后再给这个属性赋值就不会再创建新属性了，而只是重置这个属性的值：

```
objectRef.testNumber = 8;
/* - 或:- */objectRef["testNumber"] = 8;
```

每个 js 对象都有一个原型（prototype），这个原型本身也可以是一个对象，这点我们稍后会讲。该原型对象也可能有命名的属性，但是它们和赋值过程无关。赋值时，如果被赋值的对象没有对应的命名属性，则该命名属性将在该对象上被创建，并把值赋给这个新创建的属性。在此之后的赋值操作只会重置(reset)这个属性的值。

读值

在读值的过程中，原型才真正开始发挥作用。如果对象上有对应的命名属性，则该属性的值将被返回：

```
/* 给对象的一个命名属性赋值，如果在此之前对象上没有这个属性，则该属性将被创建*/objectRef.testNumber = 8;


/*读取这个属性的值:- */
var val = objectRef.testNumber; /* val 的值现在是8了 */
```

由于所有的 js 对象都有原型，而原型本身也是对象，所以原型也有原型，原型的原型也有原型。。这样就形成了原型链（prototype chain）.这个原型链会一直继续直到原型链中的某个对象的原型为 null. Object 对象的原型默认有一个 null 的原型(即 Object.prototype.prototype = null)， 因此:

```
var objectRef = new Object();
```

objectRef 的原型链中将只有一个对象，即 Object.prototype. 然而：

```
/*
   MyObject1的构造函数
*/
function MyObject1(formalParameter){
    /* 该类的对象有一个名为 'testNumber' 的命名属性  */
    this.testNumber = formalParameter;
}

/*
   MyObject2的构造函数
*/
function MyObject2(formalParameter){
   /* 该类的对象有一个名为 'testString'  的命名属性   */
    this.testString = formalParameter;
}

/* 下面的操作将所有 MyObject2类型实例的原型由默认的对象（? ? Object.prototype or Function.prototype??）替换为一个 MyObject1类型的对象，且该对象的 testNumber 属性的值为8 */MyObject2.prototype = new MyObject1( 8 );

/*
```

```
    最后，创建 MyOject2类型的一个实例,并传递一个值给它的 testString 属性 */
    var objectRef = new MyObject2( "String_Value" );
```

现在，objectRef 拥有一个原型链，该原型链上由近及远依次是 MyOject2.prototype，它是一个 testNumber 值为8的 MyObject1对象，然后是 MyObject1.prototype,这是 MyObject 类的默认原型(这是一个仅有一个 constructor 属性的对象，并且该属性的值是构造函数 MyObejct1()的引用)，然后是这个默认原型对象的原型，即 Object.prototype,由于 Object.prototype 的原型为 null，因此原型链终止。即 objectRef 的原型链为:

```
    new MyObject1( 8 ) --->
MyOject1.prototype({constructor:MyObject1(formalParameter)}) --->
Object.prototype
```

当我们试图从 **objectRef 所引用的对象(以下简称为 objectRef)**中读值时，整个原型链都会进入这个搜索过程，例如下面的例子：

```
    var val = objectRef.testString;
```

由于 objectRef 本身有一个名为 testString 的属性，因此该属性的值被返回 ("String_Value")被赋给变量 val. 然而:

```
    var val = objectRef.testNumber;
```

由于 objectRef 中并没有一个名叫 testNumber 的属性，因此 js 引擎开始检查 objectRef 的原型链。然后它在原型链的第一个对象 (new MyObject(8)) 中找到了一个名为 testNumber 的属性，因此该属性的值被返回，查找结束。下面：

```
    var val = objectRef.toString;
```

同样，objectRef 中没有名为 toString 的属性，因此 js 引擎转到原型链中查找,由于原型链第一个对象中没有相应的属性,于是它接着到第二对象中去找，也没找到，于是到第三个 (Object.prototype) 中去找，发现 Object.prototype 中有这样一个属性 (是一个函数)，于是该属性的值被返回，查找结束。最后：

```
    var val = objectRef.madeUpProperty;
```

将返回 undefined，因为 objectRef 本身以及它的原型链上的所有对象中都没有一个名叫 madeUpProperty 的属性，因此查找失败，返回 undefined.

从对象中读取某个命名属性的值时，将返回从对象本身或它的原型链中第一个找到的值；而赋值的过程则只跟这个对象本身有关，若它本身不存在这个命名属性，则将为它创建这样一个属性。

也就是说，如果我们试图进行这样的赋值操作：

```
objectRef.testNumber = 3
```

将会在 objectRef 本身中创建这样一个属性,这之后的任何读取 testNumber 值的操作都将直接返回 objectRef 本身中这个属性的值,它的原型链将不再被查找。但是它原型链中的这个属性的值不会被影响，依然是8. 只是读值时 objectRef 中的 testNumber 将原型链中的同名属性给遮住了，使它‘不可见’了。

执行上下文，作用域链 和 标识符解析

执行上下文，或运行期上下文（execution context）是 ECMAScript 规范中的一个抽象的概念，用来定义 **ECMAScript 实现**的行为准则. 但是规范并没有明确定义执行上下文应怎样被实现，除了说过执行上下文应该包含一些关联属性，这些关联属性由规范定义。因此执行上下文可以被当作（甚至是实现）为一组拥有某些属性的对象，尽管这些属性不是 public 的。(An execution context is an abstract concept used by the ECMSScript specification (ECMA 262 3rd edition) to define the behaviour required of ECMAScript implementations. The specification does not say anything about how execution contexts should be implemented but execution contexts have associated attributes that refer to specification defined structures so they might be conceived (and even implemented) as objects with properties, though not public properties.)

所有 js 代码都运行在一个执行上下文中。全局代码运行在全局执行上下文中；函数（或构造函数）的每一次调用也有相应的执行上下文。通过 eval()执行的代码也有相应的执行上下文，但由于 eval()并不常用所以我们这里不讨论它。关于执行上下文的具体细节请查阅 ECMA262规范的10.2小节。

当一个 js 函数被调用时，js 引擎进入一个执行上下文，如果在这期间另一个函数被调用了，或是该函数递归调用了自身，则另一个执行上下文将被创建，然后 js 引擎进入这个新的执行上下文，当这个函数执行完毕后，js 引擎返回之前的执行上下文，从上次中断的地方继续向下执行。这样的执行机制就形成了一个**执行上下文栈**。

当一个执行上下文被创建时，一系列的事将以固定的顺序发生：

首先，在一个函数的执行上下文中，一个“活动对象”（Activacation Object）将被创建. 这个活动对象是 ECMAScript 规范的又一个机制。它可以被认为是一个对象，因为它有一些可访问的命名属性。但它又不是一个常规意义上的对象，因为它没有 prototype，并且它不可以被直接编程访问。

其次，一个 argument 对象被创建，这是一个‘类数组’（array-like）对象，因为它的属性可以通过数字索引的方式顺序访问，并且它有一个 length 属性和 callee 属性（这点已超出我们的讨论范围，具体请参阅 ECMA262规范）。

然后，该活动对象被赋予一个名为 arguments 的属性，该属性指向第2步中被创建的 arguments 对象。

接着，该执行上下文被赋予一个作用域，该作用域是由一系列对象组成的链表。每一个函数对象都有一个内部的[[scope]]属性(这个我们稍后会详细讲解)，该[[scope]]属性的值也是一个对象链表。函数被调用时，其执行上下文被赋予的作用域，即是该函数的[[scope]]属性所指向的对象链表。并且，第一步中创建的活动对象将被添加到这个链表的顶部。（函数每次执行时都会创建一个新的活动对象，因此活动对象是不同，但作用域链表的其它部分基本是相同的，具体看下面的图吧）

接下来发生的是'**变量实例化**'(variable instantiation)，函数的所有参数、内部函数声明、局部变量和内部函数表达式（这两个是同级别的，按代码的先后顺序执行）等将依次（注意顺序，函数声明的创建在局部变量之前，因此，同名的变量将覆盖同名的函数声明，见下面的 test()，这可是一道面试题哦~~）被映射成第一步所创建的活动对象的命名属性。然后，是为函数的参数赋值。如果某个参数被传递了值，则活动对象的相应命名属性将被赋予该值，否则该参数对应的命名属性的值被赋予 undefined. 再然后，执行内部函数声明，活动对象

上对应的命名属性的值将指向这个新创建的内部函数对象。

```
    //这个例子是我加的，为了帮助大家理解函数的执行过程，非原文内容
function test() {
        var a = 1;
        function a() {
            console.log(123);
        }
        return a;
    }
    var t = test();
    console.log(typeof t); //number
```

需要注意的是，局部变量及内部函数表达式的值在'变量实例化'之后都是undefined，变量实例化只是在活动对象中创建了与它们一一对应的命名属性。这些命名属性只有在函数执行到对应的赋值语句时才会真正被赋值。见test2()：

```
    //非原文内容
function test2() {    console.log(typeof b); //function    var b = 1;
console.log(typeof b); //number
        function b() {
            console.log(123);
        }    console.log(typeof b); //number
}
test2();
```

在变量实例化之后， var b =1;执行之前，b 被映射成活动对象上的一个同名属性 b，由于此时内部函数声明已经执行完毕，而局部变量 b 尚未被赋值，因

此此时 b 的值是一个函数。而在 var b = 1;执行之后,b 被重新赋值, 又因为函数声明只在变量实例化时被执行一次，之后不会再被执行，因为在 function b()之后和之后，b 的类型都是 number. 再看下面 test3()和 test4()：

```
    //非原文内容
function test3() {
        console.log(typeof c);//undefined
        var c = 1;
        console.log(typeof c);//number
        var c = function() {}
        console.log(typeof c);//function
    }
    test3();

    function test4() {
        console.log(typeof d);//undefined
        var d = function() {}
        console.log(typeof d);//function
        var d = 1;
        console.log(typeof d);//number
    }
    test4();
```

局部变量和函数表达式的执行机制一样，因此是谁在前谁先被执行，后面的赋值将重置前面的。

最后，一个叫 this 的关键字被赋值，如果赋予给它的值是一个对象，那么 this.xx 将指向该对象中相应的属性。 如果被赋的值是 null（注意这个赋值操作是 js 引擎内部的机制，我们是无法通过编程方式控制的），则 this 将指向全

局对象。

全局执行上下文有一些特殊，因为它没有 arguments，所以它不需要定义一个活动对象来指向这个 arguments. 但是这个全局执行上下文也有作用域，只不过这个作用域链中只有一个对象，即全局对象。 另外，全局执行上下文也会执行‘变量实例化’，在这个过程中，全局对象本身充当了活动对象，这也是为什么,在全局上下文中定义的变量和函数都是全局对象的属性的原因了。全局上下文中，this 关键字指向全局对象。

作用域链(scope chain)和[[scope]]

函数调用时的作用域链，其实就是通过将活动对象添加到该函数对象的[[scope]]属性所指向的对象链表（下文简称为[[scope]]链表）的顶部形成的，因此理解内部的[[scope]]属性的定义是很重要的。

在 JS 中，函数也是对象，它们是通过函数声明在变量实例化期间被创建，或通过函数表达式的在执行期间被创建,或通过调用 Function 构造函数来创建。

通过 Function 构造器创建的函数对象， 其[[scope]]链表中永远只有一个对象——全局对象。

通过函数声明或函数表达式创建的函数对象，其[[scope]]链表指向它们被创建时的执行上下文的作用域链。 看下面的例子：

```
function foo(formalParameter){
    ...   // function body code
}
```

上面这个函数在全局执行上下文的‘变量实例化’期间被创建，因此它的[[scope]]链表指向全局作用域。由于全局作用域链表中只有一个全局对象，因此 foo()的[[scope]]链表中也只有一个全局对象。

一个相似的例子，同样是在全局环境中，以函数表达式的方式：

```
var foo = function(){
    ...   // function body code
}
```

在这个例子中，foo()的[[scope]]链表中依然只有一个全局对象，只不过名为 foo 的命名属性是在变量实例化期间添加到全局对象中,而该命名属性所对应

的函数对象是在执行期才被创建的。

由于内部函数声明和内部函数表达式所对应的函数对象是在外部函数的执行上下文中创建的，因此它们拥有更丰富的作用域链。考虑下面这个例子，在外部函数中定义了一个内部函数声明，然后执行这个外部函数：

```
function exampleOuterFunction(formalParameter){
    function exampleInnerFuncitonDec(){
        ... // inner function body
    }
    ...  // the rest of the outer function body.
}

exampleOuterFunction( 5 );
```

exampleOuterFunction()是全局上下文的’变量实例化‘过程中被创建的，因此它的作用域链等于全局执行上下文：只有一个全局对象，即exampleOuterFunction.[[scope]] = 全局对象。当全局上下文执行到exampleOuterFunction(5);这句时，一个新的执行上下文将被创建，同时被创建的还有一个活动对象。这是 exampleOuterFunction()的执行上下文，该执行上下文的作用域链由这个新创建的活动对象和 exampleOuterFunction 的[[scope]]链组成，即活动对象--->全局对象。然后是新执行上下文的变量实例化，在这个过程中，一个名为 exampleInnerFuncitonDec 的命名属性被添加到活动对象上，该属性的值是一个函数对象。该函数对象的[[scope]]链表被初始化为当前的执行上下文，即 exampleInnerFuncitonDec.[[scope]] = ”活动对象--->全局对象“。

到目前为止，一切都是由代码结构和代码执行自动控制的。**执行上下文的作用域链定义了它内部函数对象的[[scope]]链；内部函数对象的[[scope]]链又定义了它自身的执行上下文：活动对象+[[scope]]链**。但是 ECMAScript 规范提供了

with 语句用来改变函数的执行上下文。

with 语句评估一个表达式，如果这个表达式是一个对象，那个这个对象将被添加到执行上下文（是一个对象链表）的顶部(在活动对象之前). 在 with 语句块中，执行上下文将被暂时改变：在链表顶部添加了一个对象。在 with 语句执行完毕后，该对象也将从链表顶部被删除，执行上下文恢复成 with 语句之前的状态。

函数声明是不会被 with 语句影响的，因为它们是在变量实例化期间创建的(而 with 语句是在代码执行期间才被评估的)，而函数表达式却可以，因为它是在代码执行时才被创建的：



```
/* 创建一个全局变量 y,指向一个对象 */
var y = {x:5};
function exampleFuncWith(){
    var z;
    /* 将 y 所指向的对象添加到作用域链的顶部 */
    with(y){
        /*通过函数表达式创建一个函数对象,并将该对象赋值给局部变量
z */        z = function(){
            ...
        }
    }
    ...
}

/* 执行外部函数 */exampleFuncWith();
```



当 exampleFuncWith();执行时，exampleFuncWith 的执行上下文被创建，即活动对象--->全局对象。当执行到 exampleFuncWith 中的 with 语句时，全局变量 y 指向的对象被添加到执行上下文链顶部，然后 z 指向的函数对象被创建，该

函数对象的[[scope]]链被初始化为当前的执行上下文，即:
y--->活动对象--->全局对象。当 with 语句结束后，y 从执行上下文链中移除，但这不会改变 z()的[[scope]]链，它只会记住它创建时的上下文环境。

标识符解析

标识符是沿着作用域链被解析的。ECMA262规范将 this 作为一个关键字而非标识符，这并不难理解，因为 this 的值只与它所处的执行上下文有关，而与作用域链无关。标识符解析始于作用域链中的第一个对象，js 引擎检查该对象中是否有命名属性与这个标识符相同。由于作用域链是一个对象链，因此这个检查也包括检查该对象的原型链。也就是说，检查会按这样的顺序进行：作用域链中的第一个对象--->第一个对象的原型链--->第二个对象--->第二个对象的原型链--->...--->最后一个对象--->最后一个对象的原型链。直到在某个对象或它的原型链中找到对应的命名属性，则查找成功，查找终止；或一直到最后一个对象的原型链依然没有找到，则查找失败，查找终止。

对象属性的解析与标识符的解析过程一致，此时对象的属性名相当于标识符。全局对象总是处于作用域链的最末端。

由于函数被调用时，一个活动对象将被放入它的执行期上下文的作用域链的顶部，函数的所有参数、局部变量和内部函数都被映射成该活动对象的一个命名属性。因此，在函数体内，对该函数的参数、局部变量和内部函数的访问速度是最快的，它们将作为活动对象的命名属性被解析。

闭包

自动垃圾回收机制

ECMAScript 使用自动垃圾回收机制。规范并没有定义该机制的细节，因此不同的实现间可能会有一些差别，并且已知某些实现给了垃圾回收器一个非常低的优先级。但总的想法是如果一个对象不再可访问（不再有外界引用指向它），那么将成为垃圾回收器的回收目标，并将在今后的某个时刻销毁它并释放它所占用的系统资源。

通常来讲是这个样子的，当 js 引擎退出一个执行上下文时，与该执行上下文相关的作用域链、活动对象、内部函数对象及任何其它的对象等都不再可访问，因此也都成为垃圾回收的目标。

形成闭包闭包的形成是通过将一个内部函数的引用赋值给一个外部变量或外部对象的属性。例如：

```
    //闭包的例子
function exampleClosureForm(arg1, arg2){
        var localVar = 8;
        function exampleReturned(innerArg){          return ((arg1 +
arg2)/(innerArg + localVar));
   }
    /* 返回内部函数的引用 */
        return exampleReturned;
    }

    var globalVar = exampleClosureForm(2, 4);
```

现在，通过调用 exampleClosureForm() 创建的函数对象（exampleReturned）将不能被垃圾回收器回收了。因为有一个全局变量引用了它，甚至还能通过 globalVar(n)执行它呢。

事情现在有点复杂了，因为 globalVar 所引用的函数对象，即 exampleReturned()，它的[[scope]]链是它被创建时的执行上下文，也就是：exampleClosureForm 执行时的活动对象（后面简称外部活动对象1）---->全局对象。因此，外部活动对象现在也不能被垃圾回收，因为有外界引用指向它。**闭包就这样形成了。**之后调用 globalVar()时，它的执行上下文作用域链的第二个对象就是这个外部活动对象（第一个是 globalVar 自身相关的活动对象）。这个外部活动对象上的值仍然可以被读取和设置（见下面的例子），尽管创建它的执行上下文已经销毁了。

```
    //设置活动对象的属性值
```

```
function outer(){
        var a = 1;
        function inner(b){
             a += b; //a 的值依然可以被设置，尽管创建它的执行上下文已经销毁了
             console.log(a);
        }
        return inner;
    }

    var innerRef = outer();
    innerRef(2); //打印出3，是 a 的新值
```



在上面闭包的例子中，第一次调用 exampleClosureForm 时创建的外部活动对象 1 将保持 exampleClosureForm 返回时的状态，即 arg1=2,arg2=4,localVar=8,exampleReturned-->(此符号意为‘指向’)func obj. 如果 exampleClosureForm 再次被调用,例如:

```
    var secondGlobalVar = exampleClosureForm(12, 3);
```

一个新的执行上下文和新的活动对象（以下简称外部活动对象2）将被创建,同时一个新的函数对象将被返回，该函数对象拥有它独立的[[scope]]链：外部活动对象2---->全局对象. 外部活动对象2的状态为： arg1=12, arg2=3, localVar=8,

exampleReturned--->another func obj.

也就是说，exampleClosureForm 的第二次调用，形成了一个全新的闭包。这两次调用中形成的这两个独立的函数对象，分别被 globalVar 和 secondGlobalVar 引用，以下简称 globalVar() 和 secondGlobalVar()。globalVar() 和 secondGlobalVar() 均返回一个表达式：((arg1 + arg2)/(innerArg + localVar)). 该表达式中的这几个标识符是如何被解析的，对理解闭包至关重要。

现在，假设我们执行 globalVar()：

```
globalVar(2);
```

那么，一个新的执行上下文和一个新的活动对象将被创建（以下称为内部活动对象1），它只有一个命名属性：innerArg，值为2。该执行上下文的作用域为内部活动对象1---->外部活动对象1---->全局对象。

由于标识符沿着作用域链被解析，因此，((arg1 + arg2)/(innerArg + localVar))表达式中的几个标识符将沿着上面的作用域链解析。作用域链中的第一个对象是内部活动对象1，它只有一个 innerArg 属性，返回2，其它的几个标识符都是在外部活动对象1中找到的，分别是 arg1=2, arg2=4, localVar=8, 因此 globalVar(2)调用返回((2+4)/(2+8)).

接下来，执行 secondGlobalVar()：

```
secondGlobalVar(5);
```

另一个活动对象（以下称为内部活动对象2），innerArg=5, 和另一个执行上下文，作用域为：内部活动对象2---->外部活动对象2---->全局对象，被创建。与上面同样的解析方式，因此 secondGlobalVar(5)返回((12+3)/(5+8)).

再次执行 secondGlobalVar()：

```
secondGlobalVar(100);
```

新的活动对象（内部活动对象3）和新的执行上下文被创建：内部活动对象3---->外部活动对象2---->全局对象. 注意这和 secondGlobalVar(5)调用的区别：作用域链顶部的活动对象是不同的，但作用域链中第二个对象即外部活动对象2是相同的，因此无论 secondGlobalVar()被调用多少次，函数的返回表达式中 arg1, arg2, localVar 的值永远是12, 3和8.

这就是 ECMAScript 中内部函数在它的生存期内如何引用、访问外部函数的参数、局部变量、内部函数的机制。(This is how ECMAScript inner functions gain, and maintain, access to the formal parameters, declared inner functions and local variables of the execution context in which they were created. And it is how the forming of a closure allows such a function object to keep referring to those values, reading and writing to them, for as long as it continues to exist.) 该内部函数被创建时的外部活动对

象始终处于它的作用域链上，直到不再有外部引用指向该内部函数，这时，该内部函数对象将成为垃圾回收器的回收目标， 同时它作用域链上所有失去外界引用的对象（外部活动对象就属于这个范围）也将被回收。

内部函数本身也可能有内部函数，因此内部函数中也可能再返回函数进而形成更深层的闭包。每深一层，函数的作用域链中就多一个活动对象。ECMAScript 规范要求作用域链的长度是有限的，但是没说这个限度具体是多少。不同的 JS 实现（JS 引擎）可能设定了不同的长度限制但目前为止没有具体的值被公开。但这个值远远超过了你在代码中真正想要嵌套的层次。

闭包能做什么？

奇怪的是这个问题的答案似乎是任何东西，一切。闭包使得 JavaScript 可以模拟任何东西，因为使用闭包的唯一限制是你想像的力和执行力。这确实有点深奥，因此以一些比较实际的东西开始可能会比较好。

例1：给 setTimeout 传递函数引用

使用闭包的一个常见情景就是在调用一个函数之前给它传递参数。

例如，将一个函数作为参数传递给 setTimeout，使 setTimeout 在指定时间（由第二个参数指定）之后调用它，这个 js 中是很常见的。但是，这样无法给该函数传递参数（当然 setTimeout 还有另外一种用法，即第一个参数是字符串的情况，这种情况下可以将参数拼接在字符串，但这不是我们要讨论的话题）。

为了在将函数作为第一个参数传递给 setTimeout 的同时，也能给该函数传递参数，我们可以调用另一个函数，这个函数返回内部函数的一个引用，然后将该内部函数的引用作为第一个参数传递给 setTimeout. 内部函数执行时所需的参数在调用外部函数时传入。这样，setTimeout 在执行该内部函数时无需传递任何参数，但该内部函数依然可以访问外界提供的参数---这些参数是在外部函数被调用时传入的：

```
function callLater(paramA, paramB, paramC){
    /* 返回一个匿名的内部函数的引用 */
    return (function(){
        /* 这个内部函数将通过 setTimeout 被执行，当它执行时，它可以
```

```
读取、设置传递给外部函数的参数 */          paramA[paramB] = paramC;
        });
    }


    ...

    /* 调用 callLater,返回内部函数的一个引用给局部变量 funcRef。传递给外部函数 callLater 的参数将在该内部函数最终被执行时使用 */
    var functRef = callLater(elStyle, "display", "none");/* 将内部函数的引用 funcRef 作为第一个参数传递 setTimeout */hideMenu=setTimeout(functRef, 500);
```



例2：将函数作为对象的方法执行

(

There are many other circumstances when a reference to a function object is assigned so that it would be executed at some future time where it is useful to provide parameters for the execution of that function that would not be easily available at the time of execution but cannot be known until the moment of assignment.

One example might be a javascript object that is designed to encapsulate the interactions with a particular DOM element. It hasdoOnClick, doMouseOver and doMouseOut methods and wants to execute those methods when the corresponding events are triggered on the DOM element, but there may be any number of instances of the javascript object created associated with different DOM elements and the individual object instances do not know how they will be employed by the code that instantiated them. The object instances do not know how to reference themselves globally because they do not know which global variables (if any) will be assigned references to their instances.

So the problem is to execute an event handling function that has an association with a particular instance of the javascript object, and knows which method of that object to call.

The following example uses a small generalised closure based function that associates object instances with element event handlers. Arranging that the execution of the event handler calls the specified method of the object instance, passing the event object and a reference to the associated element on to the object method and returning the method's return value.

)

实话说这段话没看太懂，所以我也不勉强翻译,下面的例子我也没太明白作者想表达的意思，所以先大家自己看吧，周一我跟我们头儿探讨下,若有所得会来更新。若有哪位朋友看懂的，请留言告诉一下，不胜感激。



```
/*为指定的 dom 元素绑定事件处理程序
@param Object obj 要绑定事件处理程序的 dom 元素
@param String methodName 要作为事件处理程序的方法名称
@return Function 事件处理程序
*/
functionassociateObjWithEvent(obj, methodName){
    /* 注意: 下面返回的这个方法只有在特定事件发生时,才真正被执行*/    return (function(e){
        /* 兼容低版本 IE 和标准浏览器 */            e = e||window.event;
        /* 下面的this指向事件发生源的dom元素, 因为这个内部函数是作为该 dom 元素的事件处理程序被执行的 */
        return obj[methodName](e, this);
    });
}
```

```
    function DhtmlObject(elementId){
        var el = getElementWithId(elementId);
        if(el){
            /* 绑定事件处理程序: 将DhtmlObject的指定名称的方法作为el的某个事件处理程序 */        el.onclick = associateObjWithEvent(this, "doOnClick"); //这里的this指向当前的DhtmlObject对象
            el.onmouseover = associateObjWithEvent(this, "doMouseOver");//this同上
          el.onmouseout = associateObjWithEvent(this, "doMouseOut"); //this同上
          ...
    }
}
DhtmlObject.prototype.doOnClick = function(event, element){
    ... // doOnClick 方法体
}
DhtmlObject.prototype.doMouseOver = function(event, element){
    ... // doMouseOver 方法体
}
DhtmlObject.prototype.doMouseOut = function(event, element){
    ... // doMouseOut 方法体
}
```



这样一来，任何 DhtmlObject 对象都可以将自己与它们所感兴趣的 dom 元素（通过 elementId）结合在一起了，并且不需要知道是否有其它代码与这个 dom 元素有关联，也不需要担心会污染全局环境或与其它 DhtmlObject 对象有冲突。

例3： 将有关联关系的函数封装在一起

闭包可以用来创建作用域以便将相互关联或相互依赖的代码组织在一起，

同时最小化和其它代码意外交叉的风险。假设有这样一个函数，它的作用是构建一个字符串，但要避免重复的字符串连接操作以及不必要的中间字符串的创建（例如 var a = 'aa'; var b = 'bb' + 'cc' +a; 就会创造'bbcc'这个中间字符串.）. 因此我们可以将这些字符串片段按顺序放入一个数组，最后调用 Array.prototype.join(通过传递一个空字符串作为参数)输出结果字符串. 这个数组在这里充当了缓存的作用，如果我们在函数内部定义它，则在函数的每次调用中都会重新创建这个数组，而这是不必要的，因为每次调用中这个数组只有一小部分内容会变化.

另一个方法就是在全局上下文中定义它，然后在函数中引用它，这样这个数组只会被创建一次. 但这样做的缺点就是不利于维护和代码复用. 假设我们在别的工程中也要用到这个方法，那么当我们拷贝函数的代码时，也同时要记得拷贝函数外的这个数组的代码，并且，在新的环境中，除了考虑函数名不能和新环境中有冲突外，还要注意数组名不能冲突.

而利用闭包就可以优雅地解决这一问题. 闭包可以优雅地将数组定义和上面这个依赖它的函数封装在一起，同时不必担心会污染全局作用域或与其它代码的冲突.

解决方案就是利用自执行的函数表达式创建一个新的执行上下文，在这个上下文中定义这个数组，并返回一个内部函数，该内部函数的作用与上面说到的那个函数相同. 代码如下：



```
/*
定义一个全局变量，它指向内部函数的引用，因此，下面这个函数表达式返回的内部函数可以在全局作用域中执行.
这个内部函数返回一个 HTML 字符串，代表了一个绝对定位的 div，包裹着一个 img 元素，所有的变量都作为参数提供给函数调用
*/
var getImgInPositionedDivHtml = (function(){
    /* 将 buffAr 定义为外部函数的局部变量，它只会被创建一次(因为它是外部活动对象的一个命名属性，具体参见上文中'形成闭包'一节) */
```

```
        var buffAr = [
            '<div id="',
            '',   //index 1, DIV ID attribute
            '" style="position:absolute;top:',
            '',   //index 3, DIV top position
            'px;left:',
            '',   //index 5, DIV left position
            'px;width:',
            '',   //index 7, DIV width
            'px;height:',
            '',   //index 9, DIV height
            'px;overflow:hidden;\"><img src=\"',
            '',   //index 11, IMG URL
            '\" width=\"',
            '',   //index 13, IMG width
            '\" height=\"',
            '',   //index 15, IMG height
            '\" alt=\"',
            '',   //index 17, IMG alt text
            '\"><\/div>'    ];
        /* 返回一个内部函数对象，该函数对象会在每次调用
getImgInPositionedDivHtml( ... )时被执行 */
        return (function(url, id, width, height, top, left, altText){
            /* 为数组中对应位置的元素赋值 */        buffAr[1] = id;
            buffAr[3] = top;
            buffAr[5] = left;
            buffAr[13] = (buffAr[7] = width);
            buffAr[15] = (buffAr[9] = height);
            buffAr[11] = url;
```

```
            buffAr[17] = altText;
            /* 返回合并后的字符串 */
            return buffAr.join('');
        }); //End of 内部函数
    })(); //自执行
```



如果一个函数依赖于一个（或多个）其它函数， 但这些被这依赖的函数又不想被其它代码访问的话，那么可以用和上面相同的技巧来处理。即利用闭包将这些函数封装在一起，只暴露一个入口函数给外部调用。 这样便优雅地将一堆多函数的面向过程的代码转变成了一个封装良好的、易移植的程序单元。

其它例子

可能闭包最著名的应用之一是 Douglas Crockford 的 在 JS 对象中模拟私有实例成员。这篇文章中讲述的技巧可以扩展到各种各样的数据结构中，包括在 JS 对象中模拟静态成员。 （Probably one of the best known applications of closures is Douglas Crockford's technique for the emulation of private instance variables in ECMAScript objects. Which can be extended to all sorts of structures of scope contained nested accessibility/visibility, including the emulation of private static members for ECMAScript objects.）

闭包可以实现的应用是无止境的，理解它的工作原理是了解如何使用它的最佳指导。

意外的闭包

任何使内部函数呈现在创建它的函数之外的操作都将形成一个闭包。这使得闭包非常容易被创建，js 作者甚至可以在根本不了解闭包的情况下利用内部函数完成各种各样的任务，在这种情况下，由于没有明显迹象，他并不知道自己创建了闭包以及这样做将有什么影响。

意外地创建闭包可能产生有害的副作用，例如我们下一个章节要讲的 IE 内存泄漏问题。此外，意外闭包还可能影响代码的性能。这并不是说闭包本身会对

性能产生影响，实际上，如果正确地使用，闭包可以创建相当高效的代码。真正对效率产生影响的是内部函数。

一个常见的情况就是使用内部函数作为 dom 元素的事件处理程序。例如下面代码将用来处理 a 标签上的点击事件：

```
/* 定义一个全局变量，它的值将被添加到 a 标签的 href 属性中 */
var quantaty = 5;
/* 为指定的链接元素添加点击事件监听， 同时将全局变量添加到它的 href 属性中*/
function addGlobalQueryOnClick(linkRef){
    if(linkRef){
        /* 将一个内部函数对象赋值为 linkRef 的点击事件处理程序
        */        linkRef.onclick = function(){
            this.href += ('?quantaty='+escape(quantaty));
            return true;
        };
    }
}
```

每一次 addGlobalQueryOnClick 被调用时，都将创建一个新的函数对象，并且形成一个闭包(因为 linkRef.onclick 可以在 addGlobalQueryOnClick 之外访问，而它指向 addGlobalQueryOnClick 中的创建的那个内部函数对象)。从性能的观点来看，如果 addGlobalQueryOnClick 只被调用一两次，那么这并不是一个大问题； 但如果这个函数被执行 N 多次，那么将会创建 N 多个完全独立但却功能相同的函数对象（每个链接元素分别对应一个独立的函数对象)。

上面的代码并没有也不需要用到闭包的特性，因此生成闭包是不必要的。一个和上面的例子效果完全相同的但却高效的多的做法是，将作为事件处理程序的函数在 addGlobalQueryOnClick 之外独立定义， 然后将它的引用赋值给各个 a 元素的 onclick 属性。这样只会创建一个函数对象然后在所有的 a 元素之间共享

它的引用：

```
var quantaty = 5;

function addGlobalQueryOnClick(linkRef){
   if(linkRef){                linkRef.onclick = forAddQueryOnClick;
    }
}

function forAddQueryOnClick(){
    this.href += ('?quantaty='+escape(quantaty));
    return true;
}
```

鉴于上面第一个例子的内部函数并没有利用它自身所形成的闭包的优势，因此这种情况下，最好的做法就是不使用内部函数， 这样就不会重复创建多个完全一样的函数对象。

一个类似的例子是对象的构造函数，下面的代码并不少见：

```
functionExampleConst(param){
  //对象的方法
      this.method1 = function(){//这个赋值操作将形成一个闭包，因为this.method1可以在构造函数之外访问
          ... // method body.
      };
      this.method2 = function(){//同上，这个赋值也将形成一个闭包
          ... // method body.
      };
      this.method3 = function(){//同上，这个赋值也将形成一个闭包
```

```
            ... // method body.
        };
        /* 将参数赋值给对象的属性 */
        this.publicProp = param;
    }
```

每次调用 new ExampleConst(x)创建对象时，一组新的函数对象将被创建并分别赋值给对象的成员方法，这样，有越多的 ExampleConst 实例被创建，就有越多的函数对象被创建。

Douglas Crockford 的在 js 对象中模拟私有成员的技巧利用了闭包的特性，而如果对象的方法并没有利用到闭包的优势，那么在构造函内部给对象的方法赋值的操作将影响代码的执行效率并且会消耗更多的资源(因为会有多余的函数对象被创建).

此时，更高效的做法是只创建这些函数对象一次，然后把它们的引用分别赋值给构造函数的原型的相应属性， 这样它们就可以在构造函数所创建出的所有实例中被共享了:

```
    function ExampleConst(param){
        /* 将参数赋值给对象的属性，这个属性是每个对象独有的，即每次调用构造函数都会创建一个新的属性 */
        this.publicProp = param;
    }
    /* 通过构造函数的原型给对象添加方法 */ExampleConst.prototype.method1 = function(){
        ... // method body.
    };
    ExampleConst.prototype.method2 = function(){
        ... // method body.
    };
```

```
ExampleConst.prototype.method3 = function(){
    ... // method body.
};
```

IE 内存泄漏问题

在 IE4~6中，如果某些宿主对象间存在循环引用的话，则垃圾回收器不会回收这个循环当中的任何一个对象，从而会导致内存泄漏问题（此问题在 IE7中已解决)。这里的宿主对象指任意的 Dom 对象(包括 document 对象及它的所有后代)和 ActiveX 对象。循环链中的对象不会被回收，从而它们所占用的内存和其它系统资源也不会被释放，直到浏览器关闭。



循环引用是指两个或多个对象间的引用形成一条链，最后又指回开始的那个对象。例如对象1有一个属性指向对象2，对象2有一个属性指向对象3，而对象3又有一个属性指向对象1。 如果这个循环链中都是纯的 js 对象（即不包含 DOM 对象或 ActiveX 对象)，那么当链接之外没有引用指向它们时，这条链是被会释放的； 但如果这个链中存在任何的 DOM 对象或 ActiveX 对象，那么 IE4~6的垃圾回收器将识别不出来这是一个自引用的循环链，因此也不会释放它们，这条链中的所有对象所占的内存和资源直到浏览器关闭才能被释放.

闭包尤其容易形成循环引用。当一个函数形成闭包时，例如，被当作右值赋值给一个Dom对象的事件处理程序，并且这个Dom对象的引用存在于该闭包函数的 scope 链的一个活动对象的命名属性中时，一个闭包就形成了：DOM_Node.onevent -> function_object.[[scope]] -> scope_chain

->Activation_object.nodeRef -> DOM_Node.这是很容易的发生的（见下面的例子），并且在一个大的网站中，当每个页面中都存在很多段类似这样的代码时，系统的大部分（甚至是全部）内存将被消耗掉。



我们应足够谨慎以避免形成循环引用，当确实无法避免时，我们可以采取一定的措施来弥补,例如在 IE 的 unload 事件中将所有事件监听程序设为 null. 认识到这个问题并理解闭包和闭包的机制是避免在 IE 中引发这类问题的关键。

```
    //例:由闭包形成的循环引用
(function(){
        var b=document.body; // ← 创建 docement.body 的引用
        b.onclick=function() { // ← b.onclick 指向一个函数对象
          // 这个函数对象的[[scope]]链为: 活动对象(有一个名为 b 的命名属性)---> 全局对象. 这就形成了循环引用链, 因为: document.body.onclick--->function.[[scope]]--> 活动对象.b--->document.body
          // do something...
        };
    })();
```

原文

http://www.cnblogs.com/linfengtingyu1/p/3495070.html?utm_source=tuicool

js 实现 web 网页版台球游戏

js 桌球小游戏在线试玩地址：http://keleyi.com/game/13/

游戏截图,查看大图：



完整代码，保存到 html 文件可以试玩：

Html 代码

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>js 实现 web 页面台球游戏特效 - 柯乐义</title>
```

```
    • <link href="http://keleyi.com/game/13/css/keleyi.css"
rel="stylesheet" type="text/css">
    • <script type="text/javascript"
src="http://keleyi.com/game/13/js/keleyi.js"></script>
    • </head>
    • <body>
    • <div id="table">
    • <div id="scoreBoard"></div>
    • </div>
    • <div class="bot">
    • <div id="tips"></div>
    • <div class = "ctrl">
    • <div id="force"></div>
    • <div id="shootPos">
    • <div id="dot"></div>
    • </div>
    • </div>
    • </div>
    •
    • <div style="width:780px;margin:0px auto;clear:both;">来源：<a
href="http://keleyi.com/" target="_blank">柯乐义</a> <a
href="http://keleyi.com/game/">更多游戏</a> <a
href="http://keleyi.com/a/bjad/mroxdkos.htm" target="_blank">原文</a>
<font color="#FFFFFF">适用浏览器：IE8、360、FireFox、Chrome、Safari、
Opera、傲游、搜狗、世界之窗. </font></div>
    • </body>
    • </html>
```

原文 http://ini.iteye.com/blog/1999177?utm_source=tuicool

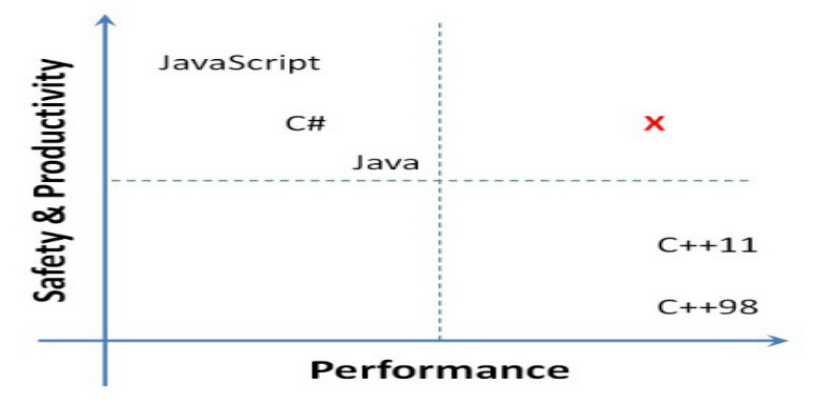
微软将推新编程语言 M#：系统编程级别的 C#

据国外媒体 VB 报道，微软正在开发一款新的编程语言：M# 。它在 C#的基础上添加了系统编程特性，可用来构建各种类型的应用，尤其是云计算应用。

微软员工 Joe Duffy 日前在其个人博客中对 M#的主要特性有：

- Lifetime understanding
- Side-effects understanding
- 大规模异步编程（Async programming at scale）
- 类型安全系统编程（Type-safe systems programming）
- 现代架构（Modern frameworks）
- 现代误差模型（Modern error model）

Joe Duffy 表示，M#相对 C#、Java 等其他语言来说，它能在“性能”、“安全与生产力”两方面会达到更好的平衡。**同时，他也表示 M#最终会开源，有可能就在令人充满期待的2014年。**他也谈到，之所以开发 M#这一新的语言，就是因为现在的编程语言太多的缘故。



一名自称前微软员工的网友在网站 Reddit 上发帖称，该语言来自 Sing#。Sing#是微软在2003到2010年研发的实验操作系统 Singularity 的系统语言。

原文

http://www.iteye.com/news/28643-Microsoft-New-Programme-Language-M-sharp?utm_source=tuicool

Spring 4 增强了对 Java 8、Java EE 7、REST 和 HTML5 的支持

几天之前 Pivotal 发布了 Spring 4.0，这是自2009年之后发布的首个主要版本。新版本支持 Java 8，支持使用 Lambda 的回调表达式、支持 JSR 310日期和时间 API 以及参数名发现。

对 Java EE 7 的支持包括：JMS 2.0、JTA 1.2、JPA 2.1、Bean 验证1.1和 JSR-236并发工具。

Spring 4还改善了 REST 支持，提供了一个新的 AsyncRestTemplate 和 HTML5/WebSocket 与旧版本浏览器的集成。

如果你想要查看 Spring 4的功能及其新特性的完整列表，那么可以查看 Pivotal 应用程序结构的首席架构师 Adrian Colyer 的博客 Spring 框架发布 4.0GA 版。

Spring 4宣称该版本能够让 Java 和 Groovy 开发者很容易地交付和整合新应用程序架构、大数据、devops 和云友好的部署。另外，它还包含一个新的名为 Spring Boot 的项目。Spring Boot 是一个新 Spring 技术，该技术的目标是让 Java 开发也能够实现 Ruby on Rails 那样的生产效率。

在 Spring 体系下有很多项目。最近的 Spring IO 项目就是为了将核心的 Spring API 带入一个有凝聚力、版本化的基础平台上而创建的。通俗地说，这是一个新的包含参考文档、教程、团队博客和论坛的网站。对于 Spring IO 和 Spring Boot 能够给开发者带来哪些帮助，Spring 4新闻稿中包含有这样一段非常有趣的话：

Spring IO 的主要目标是通过 Spring Boot 和 Groovy 这样的组件让 Java 成为开发者生产效率方面的领导者。“在 Pivotal 实验室里，开发者对幸福感和样板代码的偏执度已经形成了非常高的要求，因为他们过去完成了相当多的 Ruby on Rails 开发，” Pivotal 实验室的首席运营官 Edward Hieatt 说。“在我们的实验室里，那些使用 Spring 框架4.0和 Spring Boot 的 Ruby on Rails 开发者对它印象深刻，因为它的生产效率很高同时能够在一个单一的编程模型上处理各种各样的场景”。

我想知道 Spring 4是否是 Spring 3的一个插入式替代品，为了找到答案我

最近做了一个实验。在使用 Spring 4和 Spring Boot 对 Web 应用进行大转变一文中，我将一个使用 Spring 3.2.5、Spring 安全3.1.4和 Jersey 1.18开发的应用程序升级到了 Spring 4、Jersey 2和 Java 8。此外，我还集成了 Spring Boot 并且将其转换成了 JavaConfig 以替代 XML。整个开发体验是非常令人愉快的。我再次发现 Spring 拥有非常好的向后兼容性。而我发现的唯一问题是在缺少容器的环境中使用 Jersey 时遇到的。

在公告发布之后，我有幸采访了 Adrian Colyer。

在你说 Spring 4.0深度支持 Java 8的时候，你的意思是什么？为什么使用 Spring 能够让 Java 8更容易？

Java 8完美地契合了 Spring 的编程模型。我们相信，将 Java 8提供的那些优雅的语言特性及其立即使用的类库提升与标准的 Spring API 结合起来将会极具吸引力。事实上，Spring 框架4.0是业界第一个完全支持 Java 8的应用程序框架。为了提供能够在现在的 Java 8上开发 Spring 应用程序所需要的所有内容，我们甚至在3月份 Java 8生产启动之前就已经开始做相关的准备工作。

你提到 Spring Boot 为 Java 提供了史无前例的生产效率。那你觉得 Grails、Spring Roo、Play 和 JRebel 怎么样？它们可以让你开发一整天而不用重启应用。Spring Boot 在哪些方面要优于这些技术？

Boot 提供了比“企业”Java 更多的生产力，同时还覆盖了一套更广泛的应用程序，从 Web 到纯粹的数据应用程序（Spring XD）。而另一方面，Grails 则非常关注 Web 应用程序在完整堆栈环境中的快速开发。Spring 能处理的内容远不止 Web 应用程序，Boot 代表了可用于 Spring 及其生态系统项目的一种约定优于配置的方法。当然，我们还没有为以前所有的 Spring 项目提供 Boot 支持，但是采用的步伐是非常令人吃惊的，同时距离100%的饱和度也已经不远了。

为什么 Pivotal 会有如此之多基于 JVM 的框架（Grails、Spring Roo 以及现在的 Spring Boot）用于新应用程序的创建？

事实上，Pivotal 所有基于 JVM 的框架(Boot、Grails 和 Roo)都是基于 Spring MVC 的，但是它们的受众不同。Boot 为 Spring 生态系统提供了一种固执的、约定优于配置风格的框架，它的适用范围比 Web 应用程序更广泛。而另一方面，Grails 则更加关注领域特定的运行时，让用户能够使用 Groovy 语言和 DSL 快速

开发 Web 应用程序。最后，Roo 则是完全关注于代码生成。所有的这些框架都会将开发者带到 Spring 平台上，这是我们的主要目标。

你认为更多的选择会让开发者和决策者感到焦虑么？

这是一个棘手的或者说是比较主观的问题。对于决策者而言，更多的选择听起来好像代表着更好的风险管理，更容易招聘到开发人员。对于开发者而言，这很有可能会引发更多的焦虑，相对而言，他们只有确实找到一种比他们的背景和已有技能水平更好的风格，他们才会感到兴奋。

Spring-Loaded 看起来好像是一个免费版本的 JRebel。根据其网站提供的信息，它是 Grails 2中使用的重新装载系统。该项目与 Spring Boot 和它惊人的生产效率有关么？

不是的。在 Spring 框架4.0中 Boot 是通过@Conditional 注解启用的。你对加载的感知是准确的，但是 JRebel 更加全面，它能够和多个 IDE 一起使用。

对于 Java EE（或者其他实现）缺失的内容 Spring 4的 WebSocket 支持提供了哪些内容？

这是关于后备选项（SockJS）和高层消息协议（STOMP）的。我们主要考虑将 WebSocket 作为一个传输层，真正有趣的部分出现在它上面。Spring 的新消息模块表明了我们能够从哪里看到应用程序开发人员的共同点。它做的事情比 Java EE 7中的 JSR-356要多的多，要注意的是，我们也支持 JSR-356，它是基础，但是我们更关注自己的 WebSocket 编程模型中的更高层次的交互模式。因为低于 IE 10的浏览器并不支持 WebSocket，所以实际上如果现在没有后备选项那么 WebSocket 是无法单独使用的。

Spring 的 JSP 标签除了支持 WebSockets 和 HTML5属性之外，Spring 4还为 HTML5提供了哪些其他的特性？

我们的焦点是让开发者能够使用面向 HTML5的 Web 应用程序架构。从我们的角度，这不仅仅是通信风格和状态管理安排的问题，还包括特定的 HTML5特性。也就是说，相对于客户端上流行的 JavaScript 框架而言，Spring 也是一个好市民。我们不仅会做此时此刻的分享，我们还会致力于让 Spring 4.x 产品线持续地向下一代 Web 架构进化。

根据我的经验（来自于咨询和大会演讲），大多数开发者依然在通过 XML 使

用 Spring。你认为会有更多的开发者迁移到 JavaConfig 和 Spring 4 上么？

这是他们的选择，他们可以决定什么时候对哪些内容进行迁移对自己是有意义的。他们可以使用注解、JavaConfig 或者 XML——这比 Java EE 提供了更多的选项。顺便说一下，我们喜欢 JavaConfig，希望更多的组织能够采用它。

如果你正在使用 Spring 和 Maven，那么可以通过一个新的“材料清单”POM 进行升级。如果想要下载或者查看 Gradle 坐标，那么可以查看 Spring 框架项目的网站。

原文 http://www.infoq.com/cn/news/2006/01/spring4?utm_source=tuicool

优秀开源代码解读：RESTKit 的使用教程

这篇教程由一个独立的 iOS 开发者及4 Arrows Media 的创始人 Scott McAlister 写的。

网上有很多可用于连接和获取有趣、有用数据的 web 服务器。例如，Twitter 有一个 web 服务，您可以查看列表和发推，Foursquare 有一个 web 服务，您可以建立连接，并让服务器允许您检索你当前位置附近的餐馆的列表。如果你想要使用这些 api，您可以使用 NSURLRequest 或封装好的框架像 ASIHTTPRequest 或 AFNetworking 立即连接到他们的服务器。正如教程 other tutorials 中提到的那样。

然而，有一个更简单的方法实现这些 api——使用 RESTKit ！RESTKit 是一个易于使用的流行框架，使您不必编写很多无聊的代码，通常情况下你要当处理 web 服务 api，比如解析 JSON 或映射响应对象等。在这教程中，我们会去试试做一简单的应用程序，使用 FourSquare API 列出附近的咖啡店——因为我喜欢咖啡！

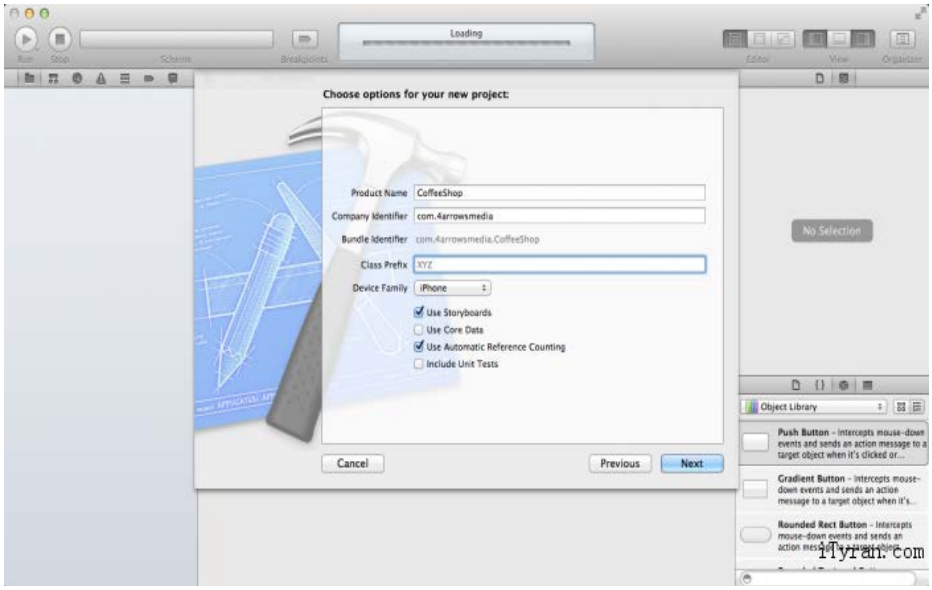
本教程是为任何水平的 iOS 开发者，从初学者到高级。请鼓起勇气读下去。

开始

首先，我们将在 Xcode 创建一个 Master-View 应用程序。如果您已经熟悉如何做到这一点，它是如何工作的，可以快速移动到下一个部分——但是我不得不为了那些初学者包括更多的细节。

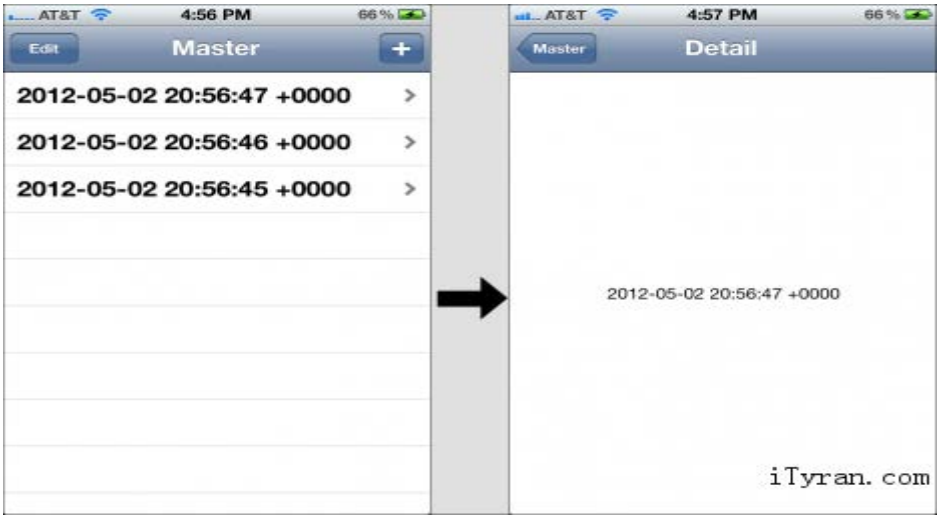
启动 Xcode 并选择 Master-Detail 程序模板创建一个新项目。输入

CoffeeShop 做为项目的名字，device family 选择为 iPhone，勾上 Use Storyboards and Use Automatic Reference Counting 复选框(其他复选框不必选)



单击 Next，并选择一个位置来保存你的项目。

选择 Master-Disciple 的，呃，等等…Master-Detail 是一个功能完备的应用程序模板，所以单击 Run 来构建和运行。您应该看到一个空白的屏幕，导航条上面有编辑和添加(+)按钮。单击“+”按钮以添加一些表条目到主屏幕。点击其中的一项将显示详细信息屏幕：



你可以看到这种格式非常适合数据列表，包含不同级别的信息。打开 MasterViewController.h：

```
#import <UIKit/UIKit.h>
```

```
•
• @interface MasterViewController : UITableViewController
•
• @end
```

MasterViewController 继承自 UITableViewController 类，管理 UITableView 并遵循 2 个协议：**UITableViewDelegate** 和 **UITableViewDataSource.**

UITableViewDataSource 是用来控制 UITableView 显示的数据，UITableViewDelegate 是用来操作 UITableView 的事件(如数据选择;置顶信息;删除和重新排序数据)

注意这些在 MasterViewController.m 里面的关键方法:

```
•
• - (NSInteger)numberOfSectionsInTableView:(UITableView *)tableView
• {
•     return 1;
• }
•
• - (NSInteger)tableView:(UITableView *)tableView numberOfRowsInSection:(NSInteger)section
• {
•     return _objects.count;
• }
•
• - (UITableViewCell *)tableView:(UITableView *)tableView cellForRowAtIndexPath:(NSIndexPath*)indexPath
• {
```

```
    • UITableViewCell *cell = [tableView dequeueReusableCellWithIdentifier:@"Cell"];
    •
    • NSDate *object = [_objects objectAtIndex:indexPath.row];
cell.textLabel.text = [object description];
    • return cell;
    • }
    •
    • - (void)prepareForSegue:(UIStoryboardSegue *)segue sender:(id)sender
    • {
    • if ([[segue identifier] isEqualToString:@"showDetail"]) {
    • NSIndexPath *indexPath = [self.tableView indexPathForSelectedRow];
    • NSDate *object = [_objects objectAtIndex:indexPath.row];
    • [[segue destinationViewController] setDetailItem:object];
    • }
    • }
    •
```

前两个方法，numberOfSectionsInTableView: 和 tableView:numberOfRowsInSection:从他们的命名可以看出来，这2个方法是告诉有表有多少分段，每个分段有多少行。

在 tableView:cellForRowAtIndexPath:方法里你可以指定每个 cell 数据的显示。

prepareForSegue:seder:方法预备在主视图（Master view）显示详情视图（Detail view）的内容。 进而用来控制视图之间的切换 。这是 storyboard 的一个特性，更多关于细节在 Beginning Storyboards in iOS 5 里可以看到。

对于本教程，您将只使用主视图控制器来显示一个列表附近的咖啡店。为简

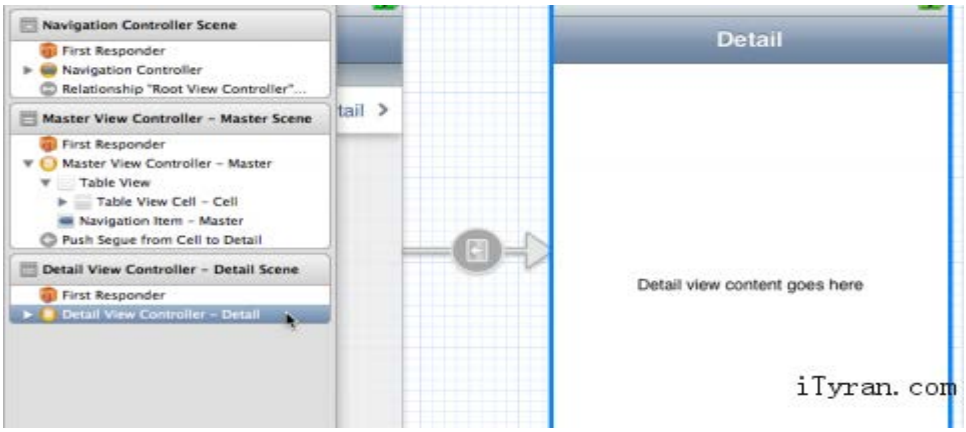
单起见,详细信息视图控制器将不被使用。所以去除如下内容:

1.MasterViewController.m 类里的 prepareForSegue:sender: 方法

2. 在 MasterViewController.m 顶部导入 #import for DetailViewController.h

3.DetailViewController.h 和 DetailViewController.m 文件

4.删除 MainStoryboard 里面的 Detail View Controller – Detail,你可以在这里找到:



RESTing Easy

现在用到有趣的东西——RESTKit ! RESTKit 主页 RestKit.org (RESTKit 的主页)描述如下:

"RestKit 是一个基于 objective – c 的 iOS 框架,其目的是为了使 RESTful web 服务更简单、快速和有趣。它基于强大的对象映射系统并且使用了一个干净、简单的 HTTP 请求/响应 API,减少完成任务所需的代码量。

RestKit 有三个主要组件:

1.Network – 在 NSURLConnection 的基础上提供了 request/response 抽象方法

2.Object Mapping – 提供了一个简单的 API 使本地对象响应远程 JSON / XML。

3.Core Data – 在对象映射(Object Map)里支持扩展。用来扩展远程资源映射到本地的对象。

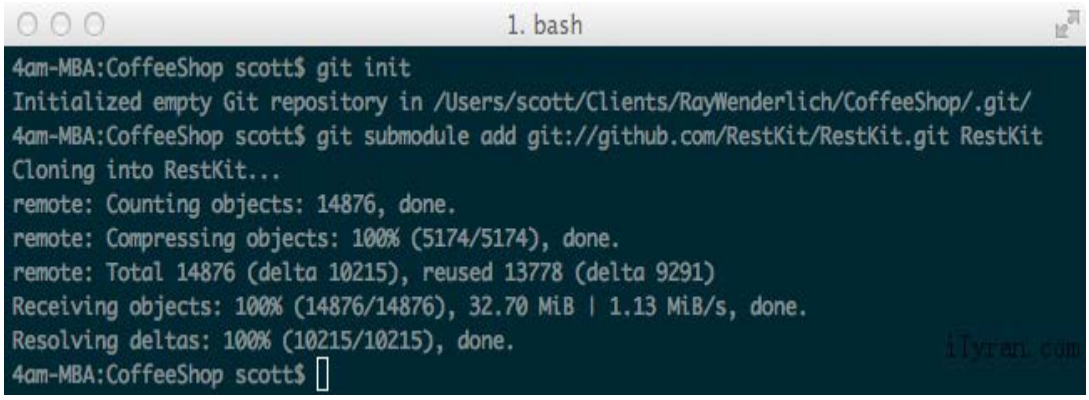
简而言之,你不必编写大量的代码!

最困难的事是安装 RestKit。当然这也不难,只是有很多的步骤而已。

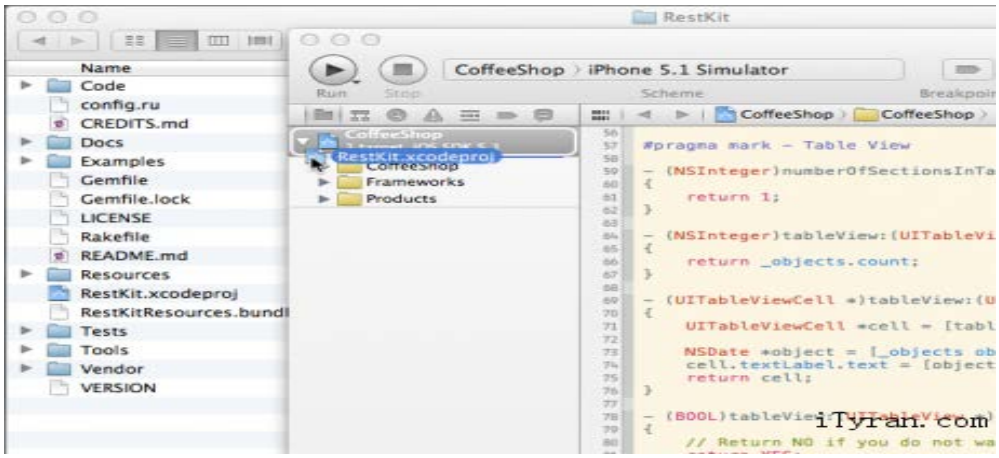
首先你要做的就是安装这个 RestKit。我推荐使用 Git 来管理你的 RestKit 安装。

打开终端,输入以下命令:

```
$ cd /path/to/MyApplication # initialize git if needed
$ git init
$ git submodule add git://github.com/RestKit/RestKit.git RestKit
```



接下来,把 RestKit 添加到您的项目 。在一个 Finder 窗口,打开 RestKit 文件夹选择你的项目文件夹。在 Xcode 里把 RestKit.xcodeproj 拖到项目导航器里 。



现在你已经告诉您的项目 RestKit 是存在的,是时候开始最关键的步骤:配置 RestKit,这样你就能正确的使用它。2个.xcodeproj 都要配置设置,添加下面这些框架来构建配置。

1.点击顶部的项目在项目导航器面板中,选择的项目的 target。

2.单击 Build Settings 在搜索框搜索 Other Linker Flags。点击 Other Linker Flags,在右边值里添加”-ObjC”。

3. 配置完之后，在搜索框里输入 Header Search Path 点击右边值里添加“$(BUILT_PRODUCTS_DIR)/../../Headers”。

4. 现在点击 Build Phases，在 Target Dependencies 点击下拉三角形。点击添加按钮，选择 REstKit。

5. 点击连接器，然后在这里选择（+）按钮添加 libRestKit.a 静态链接库。

6. 你还要添加一些苹果自带的框架，如下：（点击 cmd 可以多选）

CFNetwork.framework

CoreData.framework

libxml2.dylib

MobileCoreServices.framework

QuartzCore.framework

Security.framework

SystemConfiguration.framework

7. 还点击上次那个‘+’

设置的最后一步，验证您的 RestKit 安装和配置。打开 AppDelegate.m，并添加以下：

```
•
• #import <RestKit/RestKit.h>
```

单击运行。如果程序无错误，那么 RestKit 就设置正确了。

连接 Foursquare

现在你可以使用 RestKit 工作，你可以使用 RestKit 连接 Foursquare 的 web 服务器，为你的 app 获得“地点数据(venue data-让服务器知道你所在地点-译者注)”。

你将使用 Foursquare 的 Search Venues API 来搜索你所在地点附近的咖啡店。

别急着立刻阅读整个文档，我们只需要使用一个基本的查询而已（单击下面的连接来试一试）：https://api.foursquare.com/v2/venues/search?ll=37.33,-122.03&categoryId=4bf58dd8d48988d1e0931735，它将返回苹果总部(Apple Headquarters) 附近

的咖啡店场所的位置的 JSON 格式的数据（纬度：37.33 和 经度：-122.03）。

Foursquare 标识场所的分类是“Coffee Shop”，“categoryId=4bf58dd8d48988d1e0931735.”

对于普通查询的典型的 JSON 响应，和下面的代码相似：

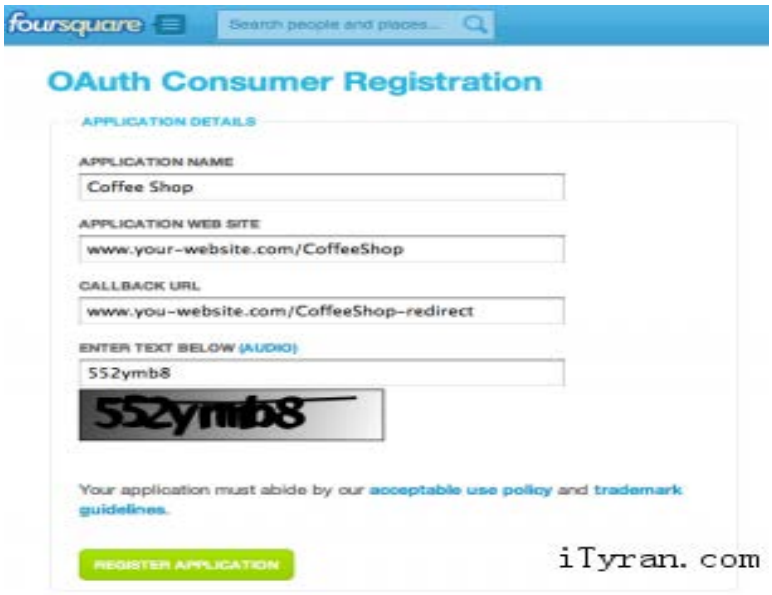
```
{
    "meta": {
        "code": 200
    },
    "notifications": [
        {
            "item": {
                "unreadCount": 0
            },
            "type": "notificationTray"
        }
    ],
    "response": {
        "venues": [
            {
                "beenHere": {
                    "count": 0
                },
                "categories": [
                    {
                        "icon": {
                            "name": ".png",
                            "prefix":
```

```
"https://Foursquare.com/img/categories/food/coffeeshop_",
                            "sizes": [
                                32,
                                44,
                                64,
                                88,
                                256
                            ]
                        },
                        "id": "4bf58dd8d48988d1e0931735",
                        "name": "Coffee Shop",
                        "pluralName": "Coffee Shops",
                        "primary": true,
                        "shortName": "Coffee Shop"
                    }
                ],
                "contact": {
                    "formattedPhone": "(650) 967-4473",
                    "phone": "6509674473",
                    "twitter": "redrockcoffee"
                },
                "hereNow": {
                    "count": 1,
                    "groups": [
                        {
                            "count": 1,
                            "items": [],
                            "name": "Other people here",
                            "type": "others"
```

```
                        }
                    ]
                },
                "id": "450af4f4f964a5204c391fe3",
                "location": {
                    "address": "201 Castro St.",
                    "city": "Mountain View",
                    "country": "United States",
                    "crossStreet": "at Villa St.",
                    "distance": 8315,
                    "lat": 37.393803003654625,
                    "lng": -122.07888155105633,
                    "postalCode": "94041",
                    "state": "CA"
                },
                "menu": {
                    "mobileUrl":
"https://Foursquare.com/v/450af4f4f964a5204c391fe3/device_menu",
                    "type": "foodAndBeverage",
                    "url":
"https://Foursquare.com/v/red-rock-coffee/450af4f4f964a5204c391fe3/me
nu"
                },
                "name": "Red Rock Coffee",
                "specials": {
                    "count": 0,
                    "items": []
                },
                "stats": {
```

```
                    "checkinsCount": 10817,
                    "tipCount": 91,
                    "usersCount": 3281
                },
                "url": "http://redrockcoffee.org/",
                "verified": true
            }
        ]
    }
}
```

Foursquare 提供免费的 web 服务访问，只要你在“OAuth Consumer Registration”页面注册你的 App。所以在继续之前，确保你完成了注册。



应用程序的 Web 站点和回调 URL 可以是您的网站上虚拟页面,因为你的应用程序将是一个移动连接和不依赖于服务器的代码。点击 Foursquare’s User Authentication 连接获取更多信息。

当你创建你的 Foursquare 应用程序注册时,你会得到一个客户机 ID 和一个客户的密码,你可以调用 Foursquare API。

你需要将这些添加打你的源代码中，这样的话当调用 Foursquare API 时，RestKit 可以验证你的应用程序。将下面的代码添

在 MasterViewController.m 中的#import 下：

```
#define kCLIENTID "Your Foursquare Client ID"
#define kCLIENTSECRET "Your Foursquare Client Secret"
```

记住你必须用从 Foursquare 中收到的真实的 Client ID 和密码取代虚拟值。

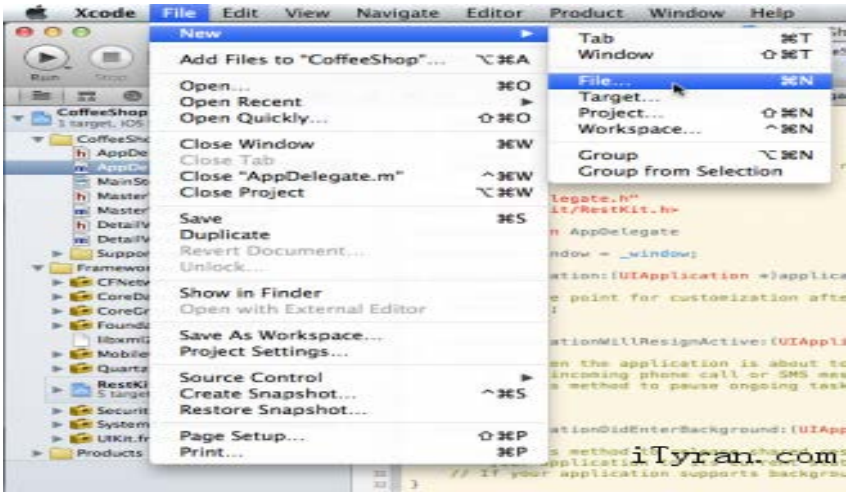
是时候写代码了

你拥有足够的代码片段可以完成你的应用程序，所以您需要做的就是添加代码！

你将使用到 RestKit 的两个组件：Network 和 Object Mapping。Network：为 Foursquare 的 API 定义基本的 URL，“https://api.foursquare.com/v2?”并发送接受你的信息。Object Mapping，你创建一个数据模型，将映射返回的 JSON 数值。

上面举出了2个创景的数据。这样开始创为基于 JSON 结构的场所数据建立数据模型类。

1. 选择 File->New->File(原来是右击 Project Navigator->New Fil，或者 command-N)



2. 选择 iOS\Cocoa Touch\Objective-C 类型，并点击 Next。

3. 输入类名例如”Venue”，选择继承于 NSObject，并点击 Next。

4. 选择文件的保存位置，并点击创建。

现在你所添加的 JSON 数据只有“场所的名字 venue name”。所以在 Venue.h 添加 name 属性：

```
@interface Venue : NSObject
```

```
•
• @property (strong, nonatomic) NSString *name;
•
• @end
```

Venue.m 中的 synthesize 属性：

```
• @implementation Venue
•
• @synthesize name;
•
• @end
```

当需要的时候，你需要继续添加。

接下来，为了 RestKit 使用修改 MasterViewController.m 添加下面的头文件：

```
• #import <RestKit/RestKit.h>
• #import "Venue.h"
```

并修改 ViewDidLoad 方法：

```
• - (void)viewDidLoad
• {
•     [super viewDidLoad];
•
•     RKURL *baseURL = [RKURL URLWithBaseURLString:@"https://api.Foursquare.com/v2"];
•     RKObjectManager *objectManager = [RKObjectManager objectManagerWithBaseURL:baseURL];
•     objectManager.client.baseURL = baseURL;
•
•     RKObjectMapping *venueMapping = [RKObjectMapping mappingForClass:[Venue class]];
•     [venueMapping mapKeyPathsToAttributes:@"name", @"name",
```

```
nil];
    • [objectManager.mappingProvider setMapping:venueMapping
forKeyPath:@"response.venues"];
    •
    • [self sendRequest];
    • }
```

这里你定义了调用 Foursquare API 基本的 URL。所有发送的请求将会添加这个基本的 URL。如果你需要再两个不同的服务器和环境(development/staging/production)之间转换的时候，这样作很好。使用基本的 URL，你创建一个 RKObjectManager 对象，RKObjectManager 是通过 HTTP 与 RESTful 服务的交互的主要接口。

RKObjectMapping 类是用来映射 JSON 数据和你的数据的模型(model)类。mapKeyPathsToAttributes 将 JSON 字段映射到您的

数据模型的属性。例如，它将 JSON 中的“id”数据 映射到 Venue 类中的 venueID 属性。

接下来，创建一个请求信息检索你的表数据，仍然在 MasterViewController.m 中，添加新的方法 sendRequest：

```
    • - (void)sendRequest
    • {
    • NSString *latLon = @"37.33,-122.03";
    • NSString *clientID = [NSString
stringWithUTF8String:kCLIENTID];
    • NSString *clientSecret = [NSString
stringWithUTF8String:kCLIENTSECRET];
    •
    • NSDictionary *queryParams;
    • queryParams = [NSDictionary
dictionaryWithObjectsAndKeys:latLon, @"ll", clientID, @"client_id",
clientSecret, @"client_secret", @"coffee", @"query", @"20120602", @"v",
```

```
nil];
    • RKObjectManager *objectManager = [RKObjectManager sharedManager];
    •
    • RKURL *URL = [RKURL URLWithBaseURL:[objectManager baseURL] resourcePath:@"/venues/search" queryParameters:queryParams];
    • [objectManager loadObjectsAtResourcePath:[NSString stringWithFormat:@"%@?%@", [URL resourcePath], [URL query]] delegate:self];
    • }
```

RestKit 将会通过 delegate methods 返回响应。为了接受这样响应，你需要实现 RKObjectLoaderDelegate 协议。

这样的话，在 MasterViewController.h 中声明：

```
    • @interface MasterViewController : UITableViewController <RKObjectLoaderDelegate>
```

然后在 MasterViewController.m 文件中，添加 RKObjectLoaderDelegate 的方法：

```
    • - (void)objectLoader:(RKObjectLoader *)objectLoader didFailWithError:(NSError *)error
    • {
    • NSLog(@"Error: %@", [error localizedDescription]);
    • }
    •
    • - (void)request:(RKRequest*)request didLoadResponse:(RKResponse*)response {
    • NSLog(@"response code: %d", [response statusCode]);
    • }
    •
    • - (void)objectLoader:(RKObjectLoader *)objectLoader
```

```
didLoadObjects:(NSArray *)objects
    • {
    •     NSLog(@"objects[%d]", [objects count]);
    •     data = objects;
    •
    •     [self.tableView reloadData];
    • }
```

唯一一个必须实现的方法是 objectLoader:didFailWithError，但是你也需要实现 objectLoader:didLoadObjects 检索你的请求数据。他返回基于在 viewDidLoad 注册的对象映射 Venue 对象数组。

如果你现在试图编译，你将得到一个报错，因为在 objectLoader:didLoadObjects 的 data 没有定义。

在 MasterViewController.m 中的修改如下：

```
    • @interface MasterViewController () {
    •     NSMutableArray *_objects;
    • }
    • @end
    •
    • @implementation MasterViewController
```

to:

```
    • @interface MasterViewController ()
    •
    • @property (strong, nonatomic) NSArray *data;
    •
    • @end
    •
    • @implementation MasterViewController
    • @synthesize data;
```

并修改：

```
- (UITableViewCell *)tableView:(UITableView *)tableView cellForRowAtIndexPath:(NSIndexPath *)indexPath
{
    UITableViewCell *cell = [tableView dequeueReusableCellWithIdentifier:@"Cell"];

    NSDate *object = [_objects objectAtIndex:indexPath.row];
    cell.textLabel.text = [object description];
    return cell;
}
```

为：

```
- (UITableViewCell *)tableView:(UITableView *)tableView cellForRowAtIndexPath:(NSIndexPath *)indexPath
{
    UITableViewCell *cell = [tableView dequeueReusableCellWithIdentifier:@"Cell"];

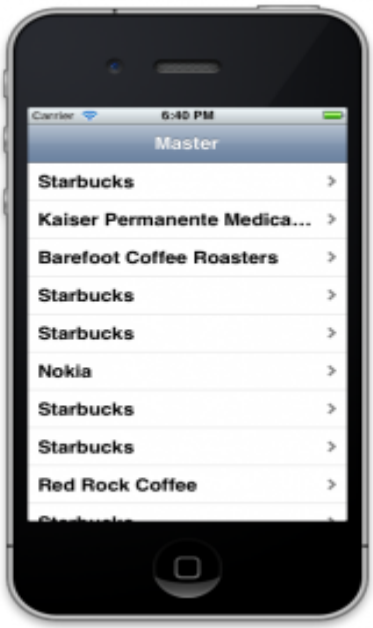
    Venue *venue = [data objectAtIndex:indexPath.row];
    cell.textLabel.text = [venue name];
    return cell;
}
```

同时，在 MasterViewController.m，将 tableView:numberOfRowsInSection:中的返回语句改为：

```
return data.count;
```

最后，确保在 MasterViewController.m 删除了 tableView:commitEditingStyle 和 insertNewObject 方法，因为数据不能再改变(另外,Xcode 扔几个编译错误,如果您不删除/改正这些方法)。

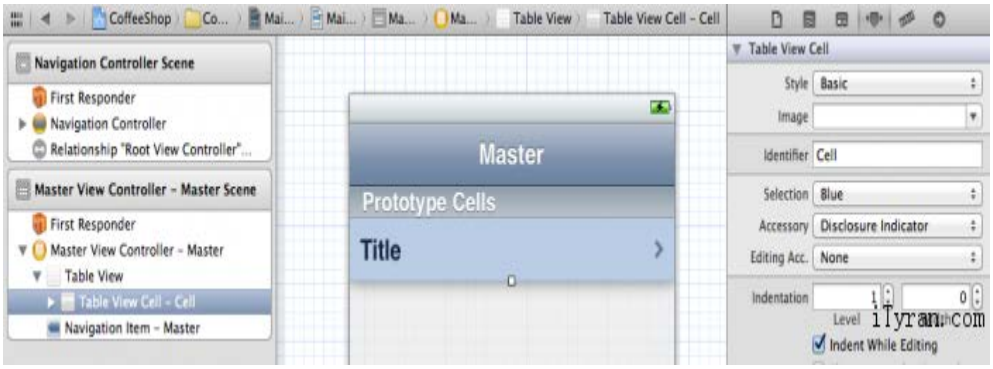
点击运行。你将看到和下图相似的情况：



你会注意到这个列表上的一些地方实际上不是咖啡馆，例如 Kaiser Permanente 和 Nokia。因为 Foursquare 数据也有坑爹的时候嘛。:P

TableViews 保留许多 TableViewCells 的集合。但由于性能的原因，它只会为可见的单元格创建足够的 TableViewCell。而不是删除和创建更多的表单元格，所以它可以循环利用那些滚动到屏幕外的表单元格。

在 tableView:cellForRowAtIndexPath 方法中，[tableView dequeueReusableCellWithIdentifier:@"Cell"]将会循环使用那些没有使用的单元格（这些表单元格的 identifier 必须和“Cell”吻合）。在 MainStoryboard.storyboard 的 Table View Cell window 中设置 identifier，如下图所示：



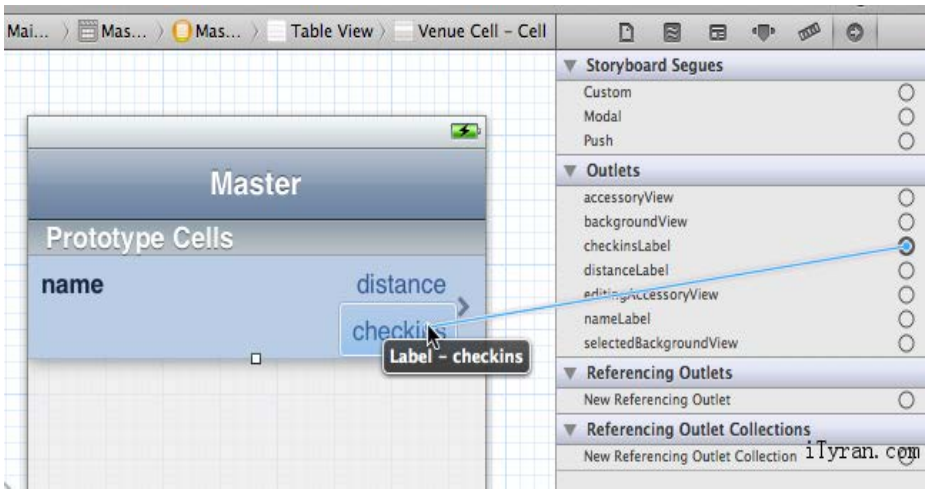
Stylin’ the TableViewCells

在上面的 storyboard 图像中另一个选项是 TableViewCell 的风格。默认是 Basic，即在 cell 中提供了一个左对齐的文本标签。其他可提供的选项是：



这些选项使你有可能在主视图中呈现你所选择的细节。比如，如果你使用一个 app 来寻找附近的咖啡店，如果我们可以看到看到每一个商店离我现在的距离，这将多么的美好。没有人会想要去很远的地方只为了一杯咖啡。

现在，使用 Right Detail style 来显示名字和距离。在下拉菜单中把原来的“Basic”风格改成“Right Detail style”



在下面的 JSON 数据中，“location”字段中就是距离

```
"location": {
        "address": "201 Castro St.",
        "city": "Mountain View",
        "country": "United States",
        "crossStreet": "at Villa St.",
        "distance": 8315,
        "lat": 37.393803003654625,
        "lng": -122.07888155105633,
        "postalCode": "94041",
        "state": "CA"
    },
```

为了使这些数据可以被 RestKit 取出，创建一个位置数据模型并提供映射。以下是步骤：

1. 新建一个文件命名为“Location”，并把它设为 NSObject 的子类。

2. 在 Location.h,增加：

```
• @property (nonatomic, strong) NSString *address;
• @property (nonatomic, strong) NSString *city;
• @property (nonatomic, strong) NSString *country;
• @property (nonatomic, strong) NSString *crossStreet;
• @property (nonatomic, strong) NSString *postalCode;
• @property (nonatomic, strong) NSString *state;
• @property (nonatomic, strong) NSNumber *distance;
• @property (nonatomic, strong) NSNumber *lat;
• @property (nonatomic, strong) NSNumber *lng;
```

请确保这些属性有在 Location.m 中被 synthesize 化！

3. 现在在 Venue.h 中增加：：

```
• #import "Location.h"
•
• @property (strong, nonatomic) Location *location;
```

在 Venue.m 中 synthesize 属性 location。

4. 在 MasterViewController.m，增加 viewDidLoad 中增加位置映射（在调用 sendRequest 之前）

```
• RKObjectMapping *locationMapping = [RKObjectMapping mappingForClass:[Location class]];
• [locationMapping mapKeyPathsToAttributes:@"address", @"address", @"city", @"city", @"country", @"country", @"crossStreet", @"crossStreet", @"postalCode", @"postalCode", @"state", @"state", @"distance", @"distance", @"lat", @"lat", @"lng", @"lng", nil];
•
• [venueMapping mapRelationship:@"location"
```

```
withMapping:locationMapping];
    • [objectManager.mappingProvider setMapping:locationMapping
forKeyPath:@"location"];
```

这和你先前做的 Venue 映射是十分相似，除了：

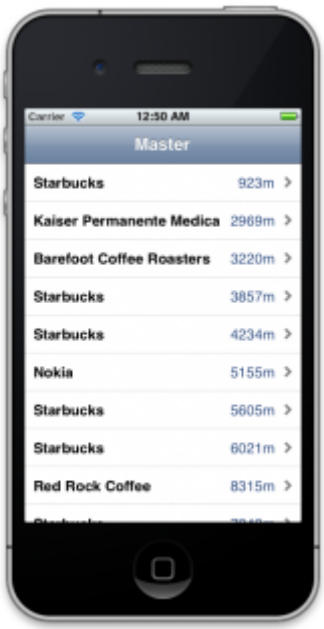
mapRelationship:withMapping:. 他通知你的地点映射实例来使用他的位置属性。

setMapping:forKeyPath: 则是告诉 objectManager 根据 JSON 数据中的位置键去使用位置映射。

5. 修改 tableView:cellForRowAtIndexPath: 为：

```
    • UITableViewCell *cell = [tableView
dequeueReusableCellWithIdentifier:@"Cell"];
    • Venue *venue = [data objectAtIndex:indexPath.row];
    • cell.textLabel.text = [venue.name length] > 24 ? [venue.name
substringToIndex:24] : venue.name;
    • cell.detailTextLabel.text = [NSString stringWithFormat:@"%.0fm",
[venue.location.distance floatValue]];
    • return cell;
```

编译运行。现在你会看见一些东西像如下图：



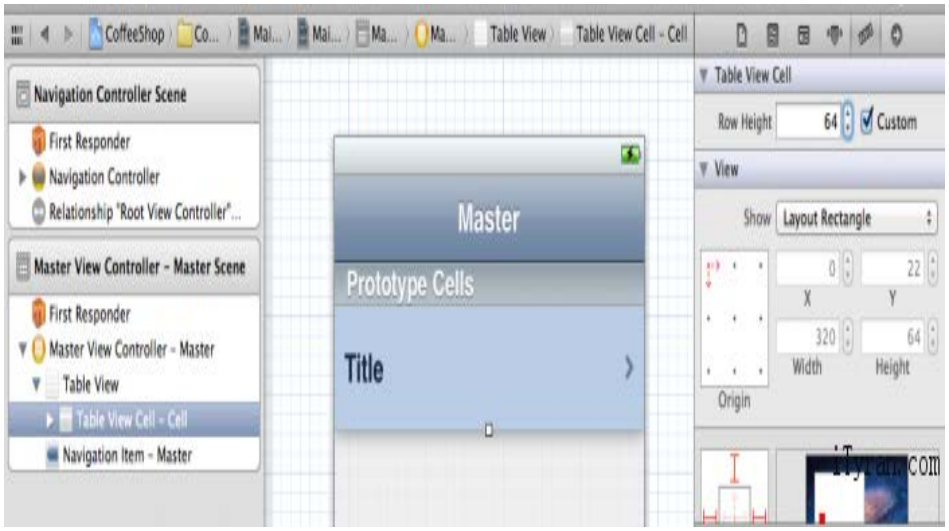
字母 m 代表的的是米，不是英里。请记得我们还在使用苹果的经纬度（译者

注：模拟器默认位置是在苹果公司总部）——如果你想你可以换成你自己的经纬度。想要获得更多怎么做的教程

自定义 TableViewCells

让我们也把 FourSquare 的 checkins 也在我们的表视图中显示出来。要做到这一点，我们需要创建一个自定义的表视图元件。

在 MainStoryboard 中，把 TableViewCell 的 style 属性改成 Custom。默认的文本标签会消失。在 Size Inspector 中改变列高度为 Custom 并从44改成64。请确保你也为 Tableview 的 rowHeight 设置成了64。



现在增加3个文本标签，就像这样：



请注意你可以使用自定义样式去纳入更多的详细信息到主屏幕。不要太多，足够就行。

因为 tableViewCell 现在被设置成了 Custom，你不能使用 UITableViewCell 的 textLabel 、detailTextLabel 属性去增加文字。这意味着为了个别标签，你需要创建一个 UITableViewCell 的子类，里面有着被 outlets 声明的 venue name, distance, and check-ins。

用 Objective-C 类模版创建一个新文件。命名为 VenueCell 并设置为 UITableViewCell 的子类。

按下面修改 VenueCell.h ：

```
#import <UIKit/UIKit.h>

@interface VenueCell : UITableViewCell

@property (strong, nonatomic) IBOutlet UILabel *nameLabel;
@property (strong, nonatomic) IBOutlet UILabel *distanceLabel;
@property (strong, nonatomic) IBOutlet UILabel *checkinsLabel;

@end
```

修改 VenueCell.m 为:

```
#import "VenueCell.h"

@implementation VenueCell
@synthesize nameLabel, distanceLabel, checkinsLabel;

@end
```

正如你所见，这个类并没有做什么：它只是增加了 nameLabel, distanceLabel、 checkinsLabel.的属性。

回到 MainStoryboard.storyboard，选择 prototype cell，设置他的类为 VenueCell。并且，在 Attributes Inspector 中，改变识别码为“VenueCell”

这些项不需要去匹配，但是最好匹配下他们为了清晰和一致性。

现在连接那些 VenueCell 中的 outlets 到文本标签。在 Connections Inspector 中，连接 Outlets：nameLabel, distanceLabel 、heckinsLabel 到他们各自的 UILabels。

在 MasterViewController.m: 的顶部增加 VenueCell 的引用

```
#import "VenueCell.h"
```

并改变 MasterViewController 中的 tableView:cellForRowAtIndex:

```
UITableViewCell *cell = [tableView
```

```
dequeueReusableCellWithIdentifier:@"Cell"];
    • Venue *venue = [data objectAtIndex:indexPath.row];
    • cell.textLabel.text = [venue.name length] > 25 ? [venue.name substringToIndex:25] : venue.name;
    • cell.detailTextLabel.text = [NSString stringWithFormat:@"%.0fm", [venue.location.distance floatValue]];
    • return cell;
```

为:

```
    • VenueCell *cell = [tableView dequeueReusableCellWithIdentifier:@"VenueCell"];
    •     Venue *venue = [data objectAtIndex:indexPath.row];
    •     cell.nameLabel.text = [venue.name length] > 25 ? [venue.name substringToIndex:25] : venue.name;
    •     cell.distanceLabel.text = [NSString stringWithFormat:@"%.0fm", [venue.location.distance floatValue]];
    •     cell.checkinsLabel.text = [NSString stringWithFormat:@"%d checkins", [venue.stats.checkins intValue]];
    •     return cell;
```

如果你尝试编译和运行这将不起作用，因为 Venue 并没有一个统计属性。正如位置数据一样，Foursquare 网站也在 JSON 数据中提供了统计数据

```
    • "stats": {
    •         "checkinsCount": 3552,
    •         "tipCount": 42,
    •         "usersCount": 1089
    •     },
```

如同位置数据一样，你需要创建一个统计数据的模型并提供映射给 RestKit

1. 创建一个新文件并命名为“Stats,”，并设置为 NSOjbect 的子类。

2. 在 Stats.h 中增加:

```
    • @property (nonatomic, strong) NSNumber *checkins;
```

- @property (nonatomic, strong) NSNumber *tips;
- @property (nonatomic, strong) NSNumber *users;

在 m 文件中 synthesize 化。

3. 在 Venue.h 中增加

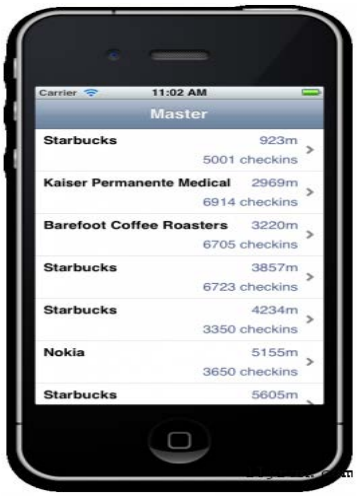
- #import "Stats.h"
-
- @property (strong, nonatomic) Stats *stats;

Venue.m 中的 synthesize 状态 .

4. 在 MasterViewController.m 的 viewDidLoad 中增加位置的映射（在 sendRequest 之前）：

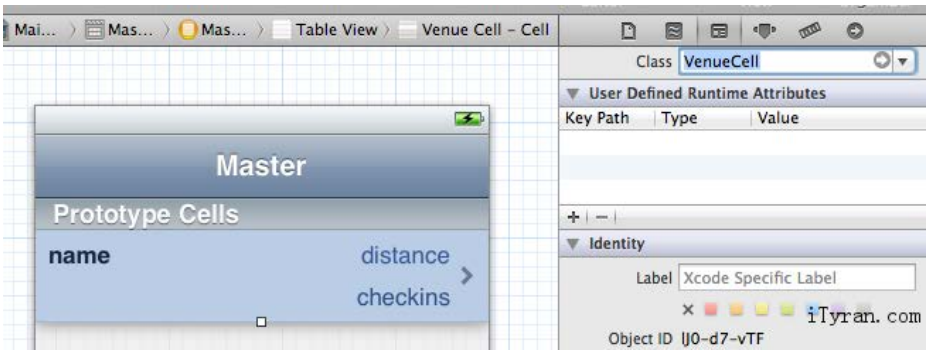
- RKObjectMapping *statsMapping = [RKObjectMapping mappingForClass:[Stats class]];
- [statsMapping mapKeyPathsToAttributes:@"checkinsCount", @"checkins", @"tipCount", @"tips", @"usersCount", @"users", nil];
-
- [venueMapping mapRelationship:@"stats" withMapping:statsMapping];
- [objectManager.mappingProvider setMapping:statsMapping forKeyPath:@"stats"];

编译运行，你可以看见像这样的东西：



谁会想到，Kaiser Permanente 竟然会被星巴克更流行！

注意：如果在这个地方你会崩，复查来确保你在MainStoryboard.storyboard中设置识别码为“VenueCell”。



何去何从？

这里给出以上教程所有的代码 。

下面给出你刚才所做的事情：

1. 用 Master-Detail Application template. 来使用 UITableView
2. RestKit 的安装和配置。
3. 为你的 app 配置 Foursquare 并介绍了他们的街道 api
4. 为你的 app 创建数据模型
5. 建立 RestKit 的映射让其可以请求，接受，解析数据进你的数据模型
6. 创建一个自定义的表视图元件来显示 Foursquare 的数据

你没有做的是：

1. 安装并使用 NSURLConnections or NSURLConnectionDelegates.
2. 解析 JSON 或 XML 数据
3. 把 JSON 或 XML 响应到你自己的对象。

因为 RestKit 都帮你做好了这些事情！不赖吧？从这里，免费设定 Foursquare 的用户认证来增加对 Foursquare API 和其例子扩展的更多特性，或找一些其他的你可以使用 RestKit 来获取数据的 web services。

原文链接

http://www.cocoachina.com/applenews/devnews/2014/0102/7641.html?utm_source=tuicool

Java 中的 Set, List, Map 漫谈

在编程语言中，集合是指代表一组对象的对象。Java 平台专门有一个集合框架(Collections Framework)。集合框架是指表示和操作集合的统一架构，隔离了集合的操作和实现细节。

集合框架中的集合接口主要分为两大部分，一部分继承自 java.util.Collection，另一部分继承自 java.util.Map （其实 Map 本质上并不是集合，只是看起来好像可以像集合一样操作)。一个有趣的事情是这些接口的实现不一定都需要实现这些接口中的修改方法（如 add，remove 等)，可以给某些不想实现的修改方法抛出一个运行时异常(UnsupportedOperationException)。

List

List 是 Java 中的一个接口，继承了 Collection 接口。它是一个有序集合，又称序列，允许存储重复元素。其实现类常用的有 ArrayList、LinkedList 等。ArrayList 是实现了 List 接口的可变长数组。它的特点是 add 方法操作时间复杂度为分期常量时间（amortized constant time)，意思即如果添加 n 个元素则耗时 O(n)，其它操作耗时则是线性时间。每个 ArrayList 都有个容量，即存放元素能力的大小。这个容量是 list 中元素个数。当添加新的元素时，这个容量也会自动添加，这需要消耗一定时间。如果要添加大量数据到 ArrayList，可以先调用 ensureCapacity 操作，从而减少每次添加新元素容量自动调整的时间。

需要注意的是 ArrayList 并不是线程同步的。如果多个线程同时访问一个 ArrayList 实例，至少一个线程修改了其结构（添加或删除元素，或显式的调整了其大小，仅仅设置元素值并不属于结构修改)，则会使程序进入不确定的状态。解决方式之一就是使用一个线程同步的对象来封装该 ArrayList。或者也可以用 Collections.synchronizedList 来封装。

```
1    List list = Collections.synchronizedList(new ArrayList(...));
```

实现原理就是Collections.synchronizedList返回的类的iterator做了特殊处理。如果iterator被创建后，除了自己的add和delete方法，有其他行为导致了List结构被修改，iterator将会抛出一个ConcurrentModificationException异常。当然iterate这种处理方式并不能担保它能处理所有的异步并发修改，只能降低程序陷入不确定状态的概率。

LikedList是一个双重链表，它既实现了List接口，也实现了Deque接口。LikedList也不是线程安全的，解决方式与ArrrayList基本相同。

Set

Set也是Java中的一个接口，同样继承于Collection。与List不同的是，Set不允许放置重复元素,并且最多只能放置一个null元素。其实现类有HashSet、TreeSet等。

HashSet的实现其实是依托了一个HashMap的实例。HashSet并不保证元素的迭代顺序每次都是一致的。HashSet的基本操作（add,remove,contains及size）耗时都是常数时间，即迭代Set的耗时与Set的大小乘以HashMap实例的乘积成正比。HashSet也不是线程安全的。

Map

Map则是另一种重要的数据结构，是一组键值对的集合。Map不允许有重复的key存在。 它的实现中有HashTable和HashMap。两者非常相似，最大的不同是HashMap不是线程安全的,并且允许null值作为key或value,而HashTable则不允许。

HashMap的性能取决于两个因素：一个是初始容量，一个是负载因数。容量是哈希表中bucket的数量。初始容量则是HashMap被创建时容量。负载因数则是当容量需要自动增加的阀值。当HashMap中的元素超过了负载因数和当前容量的乘积，HashMap则会重新进行hash计算，以便bucket数量增加到以前的近似两倍。一般负载因子的默认值是0.75，这能达到时间和空间的一个平衡。负载因子过大，虽然会减少空间消耗，但是增加查找时间。

原文链接

http://www.cnblogs.com/huang0925/p/3502929.html?utm_source=tuicool

Android 事件传递机制

实验环境

- OS X 10.9
- Eclipse(ADT)
- Android 源码版本：API Level 19（Android 4.4）

Android 事件构成

在 Android 中，事件主要包括点按、长按、拖拽、滑动等，点按又包括单击和双击，另外还包括单指操作和多指操作。所有这些都构成了 Android 中的事件响应。总的来说，所有的事件都由如下三个部分作为基础：

- 按下（ACTION_DOWN）
- 移动（ACTION_MOVE）
- 抬起（ACTION_UP）

所有的操作事件首先必须执行的是按下操作（ACTIONDOWN），之后所有的操作都是以按下操作作为前提，当按下操作完成后，接下来可能是一段移动（ACTIONMOVE）然后抬起（ACTION_UP），或者是按下操作执行完成后没有移动就直接抬起。这一系列的动作在 Android 中都可以进行控制。

我们知道，所有的事件操作都发生在触摸屏上，而在屏幕上与我们交互的就是各种各样的视图组件（View），在 Android 中，所有的视图都继承于 View，另外通过各种布局组件（ViewGroup）来对 View 进行布局，ViewGroup 也继承于 View。所有的 UI 控件例如 Button、TextView 都是继承于 View，而所有的布局控件例如 RelativeLayout、容器控件例如 ListView 都是继承于 ViewGroup。所以，我们的事件操作主要就是发生在 View 和 ViewGroup 之间，那么 View 和 ViewGroup 中主要有哪些方法来对这些事件进行响应呢？记住如下3个方法，我们通过查看 View 和 ViewGroup 的源码可以看到：

View.java

```
public boolean dispatchTouchEvent(MotionEvent event)

public boolean onTouchEvent(MotionEvent event)
```

ViewGroup.java

```
public boolean dispatchTouchEvent(MotionEvent event)

public boolean onTouchEvent(MotionEvent event)

public boolean onInterceptTouchEvent(MotionEvent event)
```

在 View 和 ViewGroup 中都存在 dispatchTouchEvent 和 onTouchEvent 方法，但是在 ViewGroup 中还有一个 onInterceptTouchEvent 方法，那这些方法都是干嘛的呢？别急，我们先看看他们的返回值。这些方法的返回值全部都是 boolean 型，为什么是 boolean 型呢，看看本文的标题，“事件传递”，传递的过程就是一个接一个，那到了某一个点后是否要继续往下传递呢？你发现了吗，“是否”二字就决定了这些方法应该用 boolean 来作为返回值。没错，这些方法都返回 true 或者是 false。在 Android 中，所有的事件都是从开始经过传递到完成事件的消费，这些方法的返回值就决定了某一事件是否是继续往下传，还是被拦截了，或是被消费了。

接下来就是这些方法的参数，都接受了一个 MotionEvent 类型的参数，MotionEvent 继承于 InputEvent，用于标记各种动作事件。之前提到的 ACTIONDOWN、ACTIONMOVE、ACTION_UP 都是 MotinEvent 中定义的常量。我们通过 MotionEvent 传进来的事件类型来判断接收的是哪一种类型的事件。到现在，这三个方法的返回值和参数你应该都明白了，接下来就解释一下这三个方法分别在什么时候处理事件。

- dispatchTouchEvent 方法用于事件的分发，Android 中所有的事件都必须经过这个方法的分发，然后决定是自身消费当前事件还是继续往下分发给子控件处理。返回 true 表示不继续分发，事件没有被消费。返回 false 则继续往下分发，如果是 ViewGroup 则分发给 onInterceptTouchEvent 进行判断是否拦截该事件。
- onTouchEvent 方法用于事件的处理，返回 true 表示消费处理当前事件，返回 false 则不处理，交给子控件进行继续分发。
- onInterceptTouchEvent 是 ViewGroup 中才有的方法，View 中没有，它的作用是负责事件的拦截，返回 true 的时候表示拦截当前事件，不继续往下分发，交给自身的 onTouchEvent 进行处理。返回 false 则不拦截，继续往下传。这是 ViewGroup 特有的方法，因为 ViewGroup 中可能还有子 View，而在 Android 中 View 中是不能再包含子 View 的（iOS 可以）。

到目前为止，Android 中事件的构成以及事件处理方法的作用你应该比较清楚了，接下来我们就通过一个 Demo 来实际体验实验一下。

Android 事件处理

首先在 Eclipse 新建一个工程，并新建一个类 RTButton 继承 Button，用来实现我们对按钮事件的跟踪。

RTButton.java

```
public class RTButton extends Button {
    public RTButton(Context context, AttributeSet attrs) {
        super(context, attrs);
    }

    @Override
    public boolean dispatchTouchEvent(MotionEvent event) {
        switch (event.getAction()) {
        case MotionEvent.ACTION_DOWN:
            System.out.println("RTButton---dispatchTouchEvent---DOWN");
            break;
        case MotionEvent.ACTION_MOVE:
            System.out.println("RTButton---dispatchTouchEvent---MOVE");
            break;
        case MotionEvent.ACTION_UP:
            System.out.println("RTButton---dispatchTouchEvent---UP");
            break;
        default:
            break;
        }
        return super.dispatchTouchEvent(event);
    }

    @Override
    public boolean onTouchEvent(MotionEvent event) {
```

```
        switch (event.getAction()) {
        case MotionEvent.ACTION_DOWN:
            System.out.println("RTButton---onTouchEvent---DOWN");
            break;
        case MotionEvent.ACTION_MOVE:
            System.out.println("RTButton---onTouchEvent---MOVE");
            break;
        case MotionEvent.ACTION_UP:
            System.out.println("RTButton---onTouchEvent---UP");
            break;
        default:
            break;
        }
        return super.onTouchEvent(event);
    }
}
```

在 RTButton 中我重写了 dispatchTouchEvent 和 onTouchEvent 方法，并获取了 MotionEvent 各个事件状态，打印输出了每一个状态下的信息。然后在 activity_main.xml 中直接在根布局下放入自定义的按钮 RTButton。

activity_main.xml

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/myLayout"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >

    <com.ryantang.eventdispatchdemo.RTButton
        android:id="@+id/btn"
```

```
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Button"/>

</LinearLayout>
```

接下来在 Activity 中为 RTButton 设置 onTouch 和 onClick 的监听器来跟踪事件传递的过程，另外，Activity 中也有一个 dispatchTouchEvent 方法和一个 onTouchEvent 方法，我们也重写他们并输出打印信息。

MainActivity.java

```
public class MainActivity extends Activity {
    private RTButton button;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        button = (RTButton)this.findViewById(R.id.btn);
        button.setOnTouchListener(new OnTouchListener() {

            @Override
            public boolean onTouch(View v, MotionEvent event) {
                switch (event.getAction()) {
                case MotionEvent.ACTION_DOWN:
                    System.out.println("RTButton---onTouch---DOWN");
                    break;
                case MotionEvent.ACTION_MOVE:
                    System.out.println("RTButton---onTouch---MOVE");
                    break;
```

```
            case MotionEvent.ACTION_UP:
                System.out.println("RTButton---onTouch---UP");
                break;
            default:
                break;
            }
            return false;
        }
    });

    button.setOnClickListener(new OnClickListener() {

        @Override
        public void onClick(View v) {
            System.out.println("RTButton clicked!");
        }
    });

}

@Override
public boolean dispatchTouchEvent(MotionEvent event) {
    switch (event.getAction()) {
    case MotionEvent.ACTION_DOWN:
      System.out.println("Activity---dispatchTouchEvent---DOWN");
      break;
    case MotionEvent.ACTION_MOVE:
         System.out.println("Activity---dispatchTouchEvent---MOVE");
      break;
```

```
            case MotionEvent.ACTION_UP:
              System.out.println("Activity---dispatchTouchEvent---UP");
              break;
            default:
              break;
            }
            return super.dispatchTouchEvent(event);
        }

        @Override
        public boolean onTouchEvent(MotionEvent event) {
            switch (event.getAction()) {
            case MotionEvent.ACTION_DOWN:
              System.out.println("Activity---onTouchEvent---DOWN");
              break;
            case MotionEvent.ACTION_MOVE:
              System.out.println("Activity---onTouchEvent---MOVE");
              break;
            case MotionEvent.ACTION_UP:
              System.out.println("Activity---onTouchEvent---UP");
              break;
            default:
              break;
            }
            return super.onTouchEvent(event);
        }
    }
```

代码部分已经完成了，接下来运行工程，并点击按钮，查看日志输出信息，我们可以看到如下结果：

Application	Tag	Text
com.ryantang.eventdispatchdemo	System.out	Activity---dispatchTouchEvent---DOWN
com.ryantang.eventdispatchdemo	System.out	RTButton---dispatchTouchEvent---DOWN
com.ryantang.eventdispatchdemo	System.out	RTButton---onTouch---DOWN
com.ryantang.eventdispatchdemo	System.out	RTButton---onTouchEvent---DOWN
com.ryantang.eventdispatchdemo	System.out	Activity---dispatchTouchEvent---UP
com.ryantang.eventdispatchdemo	System.out	RTButton---dispatchTouchEvent---UP
com.ryantang.eventdispatchdemo	System.out	RTButton---onTouch---UP
com.ryantang.eventdispatchdemo	System.out	RTButton---onTouchEvent---UP
com.ryantang.eventdispatchdemo	System.out	RTButton clicked!

通过日志输出可以看到，首先执行了 Activity 的 dispatchTouchEvent 方法进行事件分发，在 MainActivity.java 代码第55行，dispatchTouchEvent 方法的返回值是 super.dispatchTouchEvent(event)，因此调用了父类方法，我们进入 Activity.java 的源码中看看具体实现。

Activity.java

```
/**
 * Called to process touch screen events.  You can override this to
 * intercept all touch screen events before they are dispatched to the
 * window.  Be sure to call this implementation for touch screen events
 * that should be handled normally.
 *
 * @param ev The touch screen event.
 *
 * @return boolean Return true if this event was consumed.
 */
public boolean dispatchTouchEvent(MotionEvent ev) {
    if (ev.getAction() == MotionEvent.ACTION_DOWN) {
        onUserInteraction();
    }
    if (getWindow().superDispatchTouchEvent(ev)) {
        return true;
    }
    return onTouchEvent(ev);
}
```

从源码中可以看到，dispatchTouchEvent 方法只处理了 ACTIONDOWN 事件，前面提到过，所有的事件都是以按下为起点的，所以，Android 认为当 ACTIONDOWN 事件没有执行时，后面的事件都是没有意义的，所以这里首先判断 ACTION_DOWN 事件。如果事件成立，则调用了 onUserInteraction 方法，该方法可以在 Activity 中被重写，在事件被分发前会调用该方法。该方法的返回值是 void 型，不会对事件传递结果造成影响，接着会判断 getWindow().superDispatchTouchEvent(ev)的执行结果，看看它的源码：

Activity.java

```
/**
 * Used by custom windows, such as Dialog, to pass the touch screen event
 * further down the view hierarchy. Application developers should
 * not need to implement or call this.
 *
 */
public abstract boolean superDispatchTouchEvent(MotionEvent event);
```

通过源码注释我们可以了解到这是个抽象方法，用于自定义的 Window，例如自定义 Dialog 传递触屏事件，并且提到开发者不需要去实现或调用该方法，系统会完成，如果我们在 MainActivity 中将 dispatchTouchEvent 方法的返回值设为 true，那么这里的执行结果就为 true，从而不会返回执行 onTouchEvent(ev)，如果这里返回 false，那么最终会返回执行 onTouchEvent 方法，由此可知，接下来要调用的就是 onTouchEvent 方法了。别急，通过日志输出信息可以看到，ACTION_DOWN 事件从 Activity 被分发到了 RTButton，接着执行了 onTouch 和 onTouchEvent 方法，为什么先执行 onTouch 方法呢？我们到 RTButton 中的 dispatchTouchEvent 看看 View 中的源码是如何处理的。

View.java

```
/**
   * Pass the touch screen motion event down to the target view, or this
   * view if it is the target.
```

```
     *
     * @param event The motion event to be dispatched.
     * @return True if the event was handled by the view, false otherwise.
     */
    public boolean dispatchTouchEvent(MotionEvent event) {
        if (mInputEventConsistencyVerifier != null) {
            mInputEventConsistencyVerifier.onTouchEvent(event, 0);
        }

        if (onFilterTouchEventForSecurity(event)) {
            //noinspection SimplifiableIfStatement
            ListenerInfo li = mListenerInfo;
            if (li != null && li.mOnTouchListener != null && (mViewFlags &
ENABLED_MASK) == ENABLED  && li.mOnTouchListener.onTouch(this, event)) {
                return true;
            }

            if (onTouchEvent(event)) {
                return true;
            }
        }

        if (mInputEventConsistencyVerifier != null) {
            mInputEventConsistencyVerifier.onUnhandledEvent(event, 0);
        }
        return false;
    }
```

挑选关键代码进行分析，可以看代码第16行，这里有几个条件，当几个条件都满足时该方法就返回 true，当条件 li.mOnTouchListener 不为空时，通过在源码中查找，发现

mOnTouchListener 是在以下方法中进行设置的。

View.java

```
/**
 * Register a callback to be invoked when a touch event is sent to this view.
 * @param l the touch listener to attach to this view
 */
public void setOnTouchListener(OnTouchListener l) {
   getListenerInfo().mOnTouchListener = l;
}
```

这个方法就已经很熟悉了，就是我们在 MainActivity.java 中为 RTButton 设置的 onTouchListener，条件(mViewFlags & ENABLED_MASK) == ENABLED 判断的是当前 View 是否是 ENABLE 的，默认都是 ENABLE 状态的。接着就是 li.mOnTouchListener.onTouch(this, event)条件，这里调用了 onTouch 方法，该方法的调用就是我们在 MainActivity.java 中为 RTButton 设置的监听回调，如果该方法返回 true，则整个条件都满足，dispatchTouchEvent 就返回 true，表示该事件就不继续向下分发了，因为已经被 onTouch 消费了。

如果 onTouch 返回的是 false，则这个判断条件不成立，接着执行 onTouchEvent(event) 方法进行判断，如果该方法返回 true，表示事件被 onTouchEvent 处理了，则整个 dispatchTouchEvent 就返回 true。到这里，我们就可以回答之前提出的“为什么先执行 onTouch 方法”的问题了。到目前为止，ACTIONDOWN 的事件经过了从 Activity 到 RTButton 的分发，然后经过 onTouch 和 onTouchEvent 的处理，最终，ACTIONDOWN 事件交给了 RTButton 得 onTouchEvent 进行处理。

当我们的手（我这里用的 Genymotion 然后用鼠标进行的操作，用手的话可能会执行一些 ACTIONMOVE 操作）从屏幕抬起时，会发生 ACTIONUP 事件。从之前输出的日志信心中可以看到，ACTIONUP 事件同样从 Activity 开始到 RTButton 进行分发和处理，最后，由于我们注册了 onClick 事件，当 onTouchEvent 执行完毕后，就调用了 onClick 事件，那么 onClick 是在哪里被调用的呢？继续回到 View.java 的源代码中寻找。由于 onTouchEvent 在 View.java 中的源码比较长，这里就不贴出来了，感兴趣的可以自己去研究一下，通过源码阅读，我们在 ACTIONUP 的处理分支中可以看到一个 performClick()

方法，从这个方法的源码中可以看到执行了哪些操作。

View.java

```
/**
 * Call this view's OnClickListener, if it is defined.  Performs all normal
 * actions associated with clicking: reporting accessibility event, playing
 * a sound, etc.
 *
 * @return True there was an assigned OnClickListener that was called, false
 *         otherwise is returned.
 */
public boolean performClick() {
    sendAccessibilityEvent(AccessibilityEvent.TYPE_VIEW_CLICKED);

    ListenerInfo li = mListenerInfo;
    if (li != null && li.mOnClickListener != null) {
        playSoundEffect(SoundEffectConstants.CLICK);
        li.mOnClickListener.onClick(this);
        return true;
    }

    return false;
}
```

在 if 分支里可以看到执行了 li.mOnClickListener.onClick(this);这句代码，这里就执行了我们为 RTButton 实现的 onClick 方法，所以，到目前为止，可以回答前一个“onClick 是在哪里被调用的呢？”的问题了，onClick 是在 onTouchEvent 中被执行的，并且，onClick 要后于 onTouch 的执行。

到此，点击按钮的事件传递就结束了，我们结合源代码窥探了其中的执行细节，如果我们修改各个事件控制方法的返回值又会发生什么情况呢，带着这个问题，进入下一节的讨论。

Android 事件拦截

从上一节分析中，我们知道了在 Android 中存在哪些事件类型，事件的传递过程以及在源码中对应哪些处理方法。我们可以知道在 Android 中，事件是通过层级传递的，一次事件传递对应一个完整的层级关系，例如上节中分析的 ACTIONDOWN 事件从 Activity 传递到 RTButton，ACTIONUP 事件也同样。结合源码分析各个事件处理的方法，也可以明确看到事件的处理流程。

之前提过，所有事件处理方法的返回值都是 boolean 类型的，现在我们来修改这个返回值，首先从 Activity 开始，根据之前的日志输出结果，首先执行的是 Activity 的 dispatchTouchEvent 方法，现在将之前的返回值 super.dispatchTouchEvent(event)修改为 true，然后重新编译运行并点击按钮，看到如下的日志输出结果。

Tag	Text
System.out	Activity---dispatchTouchEvent---DOWN
System.out	Activity---dispatchTouchEvent---UP

可以看到，事件执行到 dispatchTouchEvent 方法就没有再继续往下分发了，这也验证了之前的说法，返回 true 时，不再继续往下分发，从之前分析过的 Activity 的 dispatchTouchEvent 源码中也可知，当返回 true 时，就没有去执行 onTouchEvent 方法了。

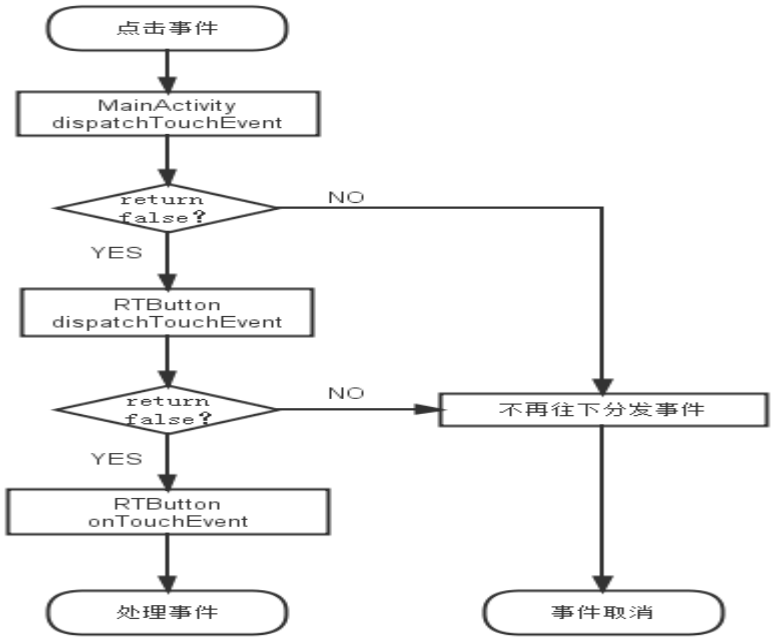
接着，将上述修改还原，让事件在 Activity 这继续往下分发，接着就分发到了 RTButton，将 RTButton 的 dispatchTouchEvent 方法的返回值修改为 true，重新编译运行并查看输出日志结果。

Tag	Text
System.out	Activity---dispatchTouchEvent---DOWN
System.out	RTButton---dispatchTouchEvent---DOWN
System.out	Activity---dispatchTouchEvent---UP
System.out	RTButton---dispatchTouchEvent---UP

从结果可以看到，事件在 RTButton 的 dispatchTouchEvent 方法中就没有再继续往下分发了。接着将上述修改还原，将 RTButton 的 onTouchEvent 方法返回值修改为 true，让其消费事件，根据之前的分析，onClick 方法是在 onTouchEvent 方法中被调用的，事件在这被消费后将不会调用 onClick 方法了，编译运行，得到如下日志输出结果。

Tag	Text
System.out	Activity---dispatchTouchEvent---DOWN
System.out	RTButton---dispatchTouchEvent---DOWN
System.out	RTButton---onTouch---DOWN
System.out	RTButton---onTouchEvent---DOWN
System.out	Activity---dispatchTouchEvent---UP
System.out	RTButton---dispatchTouchEvent---UP
System.out	RTButton---onTouch---UP
System.out	RTButton---onTouchEvent---UP

跟分析结果一样，onClick 方法并没有被执行，因为事件在 RTButton 的 onTouchEvent 方法中被消费了。下图是整个事件传递的流程图。



到目前为止，Android 中的事件拦截机制就分析完了。但这里我们只讨论了单布局结构下单控件的情况，如果是嵌套布局，那情况又是怎样的呢？接下来我们就在嵌套布局的情况下对 Android 的事件传递机制进行进一步的探究和分析。

Android 嵌套布局事件传递

首先，新建一个类 RTLayout 继承于 LinearLayout，同样重写 dispatchTouchEvent 和 onTouchEvent 方法，另外，还需要重写 onInterceptTouchEvent 方法，在文章开头介绍过，这个方法只有在 ViewGroup 和其子类中才存在，作用是控制是否需要拦截事件。这里不要和 dispatchTouchEvent 弄混淆了，后者是控制对事件的分发，并且后者要先执

行。

那么，事件是先传递到 View 呢，还是先传递到 ViewGroup 的？通过下面的分析我们可以得出结论。首先，我们需要对工程代码进行一些修改。

RTLayout.java

```
public class RTLayout extends LinearLayout {
    public RTLayout(Context context, AttributeSet attrs) {
        super(context, attrs);
    }

    @Override
    public boolean dispatchTouchEvent(MotionEvent event) {
        switch (event.getAction()) {
        case MotionEvent.ACTION_DOWN:
          System.out.println("RTLayout---dispatchTouchEvent---DOWN");
          break;
        case MotionEvent.ACTION_MOVE:
          System.out.println("RTLayout---dispatchTouchEvent---MOVE");
          break;
        case MotionEvent.ACTION_UP:
          System.out.println("RTLayout---dispatchTouchEvent---UP");
          break;
        default:
          break;
        }
        return super.dispatchTouchEvent(event);
    }

    @Override
    public boolean onInterceptTouchEvent(MotionEvent event) {
```

```
    switch (event.getAction()) {
    case MotionEvent.ACTION_DOWN:
        System.out.println("RTLayout---onInterceptTouchEvent---DOWN");
        break;
    case MotionEvent.ACTION_MOVE:
        System.out.println("RTLayout---onInterceptTouchEvent---MOVE");
        break;
    case MotionEvent.ACTION_UP:
        System.out.println("RTLayout---onInterceptTouchEvent---UP");
        break;
    default:
        break;
    }
    return super.onInterceptTouchEvent(event);
}

@Override
public boolean onTouchEvent(MotionEvent event) {
    switch (event.getAction()) {
    case MotionEvent.ACTION_DOWN:
        System.out.println("RTLayout---onTouchEvent---DOWN");
        break;
    case MotionEvent.ACTION_MOVE:
        System.out.println("RTLayout---onTouchEvent---MOVE");
        break;
    case MotionEvent.ACTION_UP:
        System.out.println("RTLayout---onTouchEvent---UP");
        break;
    default:
```

```
            break;
        }
        return super.onTouchEvent(event);
    }
}
```

同时，在布局文件中为 RTButton 添加一个父布局，指明为自定义的 RTLayout，修改后的布局文件如下。

activity_main.xml

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >

    <com.ryantang.eventdispatchdemo.RTLayout
        android:id="@+id/myLayout"
        android:layout_width="match_parent"
        android:layout_height="match_parent" >

        <com.ryantang.eventdispatchdemo.RTButton
            android:id="@+id/btn"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:text="Button" />
    </com.ryantang.eventdispatchdemo.RTLayout>

</LinearLayout>
```

最后，我们在 Activity 中也为 RTLayout 设置 onTouch 和 onClick 事件，在 MainActivity 中添加如下代码。

MainActivity.java

```
rtLayout.setOnTouchListener(new OnTouchListener() {


        @Override
        public boolean onTouch(View v, MotionEvent event) {
            switch (event.getAction()) {
            case MotionEvent.ACTION_DOWN:
                System.out.println("RTLayout---onTouch---DOWN");
                break;
            case MotionEvent.ACTION_MOVE:
                System.out.println("RTLayout---onTouch---MOVE");
                break;
            case MotionEvent.ACTION_UP:
                System.out.println("RTLayout---onTouch---UP");
                break;
            default:
                break;
            }
            return false;
        }
    });


  rtLayout.setOnClickListener(new OnClickListener() {


        @Override
        public void onClick(View v) {
            System.out.println("RTLayout clicked!");
        }
    });
```

代码修改完毕后，编译运行工程，同样，点击按钮，查看日志输出结果如下：

Tag	Text
System.out	Activity---dispatchTouchEvent---DOWN
System.out	RTLayout---dispatchTouchEvent---DOWN
System.out	RTLayout---onInterceptTouchEvent---DOWN
System.out	RTButton---dispatchTouchEvent---DOWN
System.out	RTButton---onTouch---DOWN
System.out	RTButton---onTouchEvent---DOWN
System.out	Activity---dispatchTouchEvent---UP
System.out	RTLayout---dispatchTouchEvent---UP
System.out	RTLayout---onInterceptTouchEvent---UP
System.out	RTButton---dispatchTouchEvent---UP
System.out	RTButton---onTouch---UP
System.out	RTButton---onTouchEvent---UP
System.out	RTButton clicked!

从日志输出结果我们可以看到，嵌套了 RTLayout 以后，事件传递的顺序变成了 Activity->RTLayout->RTButton，这也就回答了前面提出的问题，Android 中事件传递是从 ViewGroup 传递到 View 的，而不是反过来传递的。

从输出结果第三行可以看到，执行了 RTLayout 的 onInterceptTouchEvent 方法，该方法的作用就是判断是否需要拦截事件，我们到 ViewGroup 的源码中看看该方法的实现。

ViewGroup.java

```
public boolean onInterceptTouchEvent(MotionEvent ev) {
      return false;
   }
```

该方法的实现很简单，只返回了一个 false。那么这个方法是在哪被调用的呢，通过日志输出分析可知它是在 RTLayout 的 dispatchTouchEvent 执行后执行的，那我们就进到 dispatchTouchEvent 源码里面去看看。由于源码比较长，我将其中的关键部分截取出来做解释说明。

ViewGroup.java

```
            // Check for interception.
            final boolean intercepted;
            if (actionMasked == MotionEvent.ACTION_DOWN
                  || mFirstTouchTarget != null) {
               final boolean disallowIntercept = (mGroupFlags &
```

```
FLAG_DISALLOW_INTERCEPT) != 0;

                if (!disallowIntercept) {

                    intercepted = onInterceptTouchEvent(ev);

                    ev.setAction(action); // restore action in case it was changed

                } else {

                    intercepted = false;

                }

            } else {

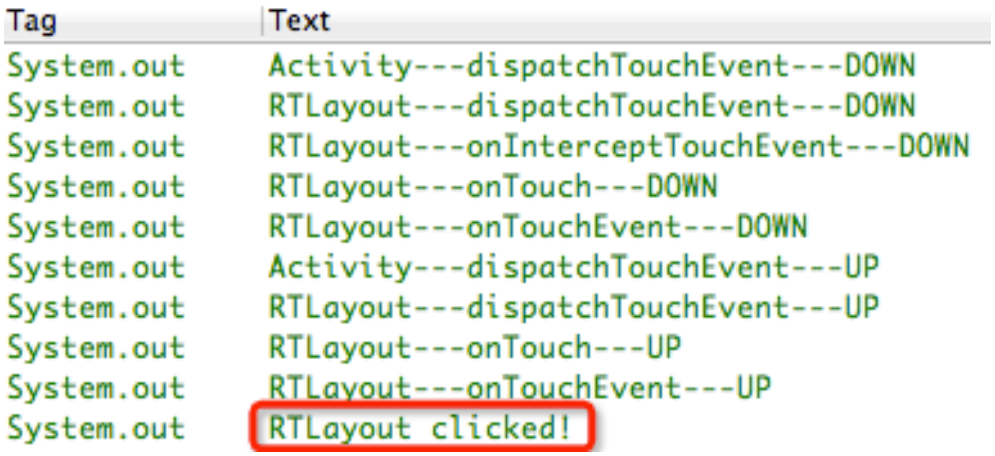
                // There are no touch targets and this action is not an initial down

                // so this view group continues to intercept touches.

                intercepted = true;

            }
```

从这部分代码中可以看到 onInterceptTouchEvent 调用后返回值被赋值给 intercepted，该变量控制了事件是否要向其子控件分发，所以它起到拦截的作用，如果 onInterceptTouchEvent 返回 false 则不拦截，如果返回 true 则拦截当前事件。我们现在将 RTLayout 中的该方法返回值修改为 true，并重新编译运行，然后点击按钮，查看输出结果如下。



Tag	Text
System.out	Activity---dispatchTouchEvent---DOWN
System.out	RTLayout---dispatchTouchEvent---DOWN
System.out	RTLayout---onInterceptTouchEvent---DOWN
System.out	RTLayout---onTouch---DOWN
System.out	RTLayout---onTouchEvent---DOWN
System.out	Activity---dispatchTouchEvent---UP
System.out	RTLayout---dispatchTouchEvent---UP
System.out	RTLayout---onTouch---UP
System.out	RTLayout---onTouchEvent---UP
System.out	RTLayout clicked!

可以看到，我们明明点击的按钮，但输出结果显示 RTLayout 点击事件被执行了，再通过输出结果分析，对比上次的输出结果，发现本次的输出结果完全没有 RTButton 的信息，没错，由于 onInterceptTouchEvent 方法我们返回了 true，在这里就将事件拦截了，所以他不会继续分发给 RTButton 了，反而交给自身的 onTouchEvent 方法执行了，理所当然，最后执行的就是 RTLayout 的点击事件了。

总结

以上我们对 Android 事件传递机制进行了分析，期间结合系统源码对事件传递过程中的处理情况进行了探究。通过单布局情况和嵌套布局情况下的事件传递和处理进行了分析，现总结如下：

- Android 中事件传递按照从上到下进行层级传递，事件处理从 Activity 开始到 ViewGroup 再到 View。
- 事件传递方法包括 dispatchTouchEvent、onTouchEvent、onInterceptTouchEvent，其中前两个是 View 和 ViewGroup 都有的，最后一个是只有 ViewGroup 才有的方法。这三个方法的作用分别是负责事件分发、事件处理、事件拦截。
- onTouch 事件要先于 onClick 事件执行，onTouch 在事件分发方法 dispatchTouchEvent 中调用，而 onClick 在事件处理方法 onTouchEvent 中被调用，onTouchEvent 要后于 dispatchTouchEvent 方法的调用。

原文链接

http://www.infoq.com/cn/articles/android-event-delivery-mechanism?utm_source=tuicool

Android 热点回顾第一期

文章、教程

高效、低延迟的 Android 音频应用开发

Nils Schneider 开发了 Heat Synthesizer 这款音乐软件。在随后发表的博客中，他提到了在处理高性能、低延迟音频时遇到的问题。他阐述了几种实现最佳性能的方法。尽管如此，仍然存在一些其他问题有待解决。

使用 Retrofit 和 Mockito 测试 Android API 的可靠性

Matt Swanson 发布了一篇优秀的博客，讨论了如何在 API 中调试一些像 Http 请求这样的棘手事件。他的办法是使用 Retrofit 库调用 REST API，然后用 Mockito 工具模拟结果。

小窍门：在滚动视图中使用 WebView（链接邮件程序）

这是 Pedro Loureiro 从 AOSP 邮件应用程序中获得的开发灵感。

AndroidStaggeredGrid — Android 瀑布流布局

瀑布流布局可实现纵横交错的多栏布局。

ART 运行时中的 OAT（文件格式）

这是 Android 最新版本（巧克力机器人）介绍系列短文中的第二篇。Matthias Schaff 将带领我们仔细了解最新 Android 版本搭载的 OAT 文件格式以及运行时垃圾信息处理机制。

使用 Mirror API 和 GDK 开发 Google 智能眼镜

让我们从开发者的视角仔细了解 Google Glass GDK。

从 ActionBarSherlock 迁移到 ActionBarCompat

在这篇文章里 Wolfram Rittmeyer 做了非常详尽的介绍，如何从“气数将尽”的 ActionBarSherlock 迁移 Google 的 ActionBarCompat 控件。

Android 平台内存溢出报错：报错原因、解决方案、最佳实践

这篇文章通过一些具体的例子介绍了如何从你的应用程序中找出那些令人厌烦的内存溢出问题。

设定受限配置（Restricted Profiles）

昨天 Netflix 为我们介绍了如何使用 Jelly Bean 配的受限置功能。尽管功能非常强大，但很少有应用在使用它。本文中，Gabriele Mariotti 将为我们演示如何在自己的应用程序中实现受限配置。

智能手表 — Part2

Mark Allison 将继续在他的文章的中向我们介绍，如何为智能手表开发 Android 应用程序。

设计

Android 实物模型（Mockup）和设计工具

Nikolai Prettner 分享了一个实体模型工具，可以用做概念验证和模型设计。此工具不仅包含了各种样式的 Google 应用，还涵盖了最新版本——4.4 Kitkat 风格的应用。

函数库、代码

Dagger 1.2版本发布

Dagger 发布了1.2版本，可以从 Maven Center 上下载。新版本对编译器和运行时做了许多性能上的提升，例如更高效的字符串连接操作、在全局锁的基础上支持读、写锁。

Calligraphy

你是否对设置自定义控件字体和遍历众多控件寻找 Textview 而感到厌烦？Christopher Jenkins 也为此感到困扰。所以他最近发布了一个函数库，简化了在 Android 应用中使用自定义字体。

DynamicShareActionProvider

这是 Google 提供的 ShareActionProvider 类的一个自定义实现——它可以动态共享数据！也就是说，在共享数据前你可以先确定共享的数据类型，这样在创建 Activity 事件时就不必定义多余的数据类型。

工具

Android Studio 0.4.0发布

新版本的支持 Android Gradle 0.7.0插件、导入 Eclipse ADT 工程和一个图像资源导入向导。这是一款用于 Android Studio 的插件，与 android-holo-colors.com 功能类似：首先为你的主题命名，选择 minSDKVersion 和 compact library（可根据需要选择）。

原文 http://www.importnew.com/8246.html?utm_source=tuicool

iOS 是怎么"绘画"的?

什么是绘图引擎

绘图引擎，通俗来说就好比给你一张纸一支笔和若干颜色的颜料，你可以用它来做**最基本**的图形绘制.

一个**最基本的绘图引擎**包括一下接口:

```
//Code-1
I1. drawLine()      //绘制任意线条,并支持对线条上色.
I2. drawPath()      //根据路径绘制形状,并支持填充颜色.
I3. drawImage()     //绘制图像(e.g. xxx.jpg, xxx.png)
I4. drawGradient()  //绘制渐变填充
I5. transform()     //矩阵映射变换.
I6. drawText()      //绘制文字
```

不难想象，有了以上接口，我们就可以**方便**的绘制任意想要的图像.

这里强调的是**方便**，有些接口并不是**必须**的. 比如说 `drawImage()`,我们总可以

调用有限次 drawLine() 和 drawShape() 来绘制任意给定的 Image. 但是复杂程度可想而知.

一个绘图引擎设计的目的就是为了方便上层调用, 所以它会封装一些**最常用**和**最基本**的接口. 以上5个接口就满足这两个条件之一. 所谓**最常用**和**最基本**并没有一个明确的定义, 所以不同的绘图引擎可能会多一些常用接口,但都大同小异.

iOS 的绘图引擎

下面我们就 Code-1里提到的接口在 iOS 平台上做一个介绍.

在哪里绘制?

如果我们在 XCode 里新建一个 UIView 类, 我们会得到以下代码:

```
//Code-2
#import "GraphicsViewControllerView.h"
@implementation GraphicsViewControllerView
- (id)initWithFrame:(CGRect)frame
{
    self = [super initWithFrame:frame];
    if (self) {
        // Initialization code }
    return self;
}
- (void)drawRect:(CGRect)rect
{
    // Drawing code
}
@end
```

通常,drawRect() 都会被注释起来. 因为, 如果你向 UIView 添加 subView 或者设置 UIView 的显示相关的属性(e.g. backgroundCrolor)的时候, UIKit 会自动的把这些参数代表的含义绘制到 CALayer 上去. 也就是说, 一般情况我们并不需要自己来绘制, UIKit 会自动帮我们完成绘制工作.

但是，当不添加 subView，不设置 UIView 的显示相关的属性时，我们就可以通过重载 drawRect() 来手动绘制图像了.

Context

A graphical context can be thought of as a canvas, offering an enormous number of properties such as pen color, pen thickness, etc. Given the context, you can start painting straight away inside the drawRect: method, and Cocoa Touch will make sure that the attributes and properties of the context are applied to your drawings. We will talk about this more later, but now, let's move on to more interesting subjects.

drawText

我们新建一个 UIView 类 CustomUIView，如下重载 drawRect() 方法.

新建 CustomUIView 的对象，不设置任何属性，添加到显示列表.

```
//Code-3
- (void)drawRect:(CGRect)rect{
    UIFont *font = [UIFont systemFontWithSize:40.f];
    NSString *myString = @"Some String";
    [myString drawAtPoint:CGPointMake(40, 180) withFont:font];
}
```

运行，就可以看到我们没有添加任何 UITextField 却显示了文字~

more:

关于文字绘制的方法还有 drawInRect:withFont: 等几个方法，可参考官方文档.

setColor

我们把 Code-3 中的代码添加两行，变成:

```
//Code-4
- (void)drawRect:(CGRect)rect{
    UIColor* color = [UIColor blueColor];       //create color
    [color set];                                //set color
```

```
        UIFont *font = [UIFont systemFontWithSize:40.f];
        NSString *myString = @"Some String";
        [myString drawAtPoint:CGPointMake(40, 180) withFont:font];
    }
```

就可以看到,文字由黑色变成了蓝色.

more:

`UIColor` 还有两个方法 `setStroke` 和 `setFill` 分别设置线条颜色和填充颜色. 在后面的章节会用到. `set` 方法影响所有前景色.

drawImage

同样的, 如下重写 `drawRect` 方法:

```
//Code-5
- (void)drawRect:(CGRect)rect{
    /* Assuming the image is in your app bundle and we can load it
*/
    UIImage *xcodeIcon = [UIImage imageNamed:@"filename.png"];
    [xcodeIcon drawAtPoint:CGPointMake(0.0f, 20.0f)];
}
```

我们看到图像被绘制出来了.

more:

`UIImage` 还有其他绘制方法:

```
drawInRect:
drawAsPatternInRect:
drawAtPoint:blendMode:alpha:
drawInRect:blendMode:alpha:
```

方法名已经很清楚的说明了方法的用途. 具体可参考官方文档

drawLine

这两个是绘图引擎里最基本的, 所以放在一起讲述.

```
//Code-6: drawLine
```

```
- (void)drawRect:(CGRect)rect{

    /* Step1 设置绘图颜色 */
    [[UIColor brownColor] set];

    /* Step2 获取当期的画布: Graphic Context */
    CGContextRef currentContext = UIGraphicsGetCurrentContext();

    /* Step3 设置线条宽度 */
    CGContextSetLineWidth(currentContext,5.0f);

    /* Step4 把画笔移动到起始点 */
    CGContextMoveToPoint(currentContext,50.0f, 10.0f);

    /* Step5 从起始点绘制线条到终点 */
    CGContextAddLineToPoint(currentContext,100.0f, 200.0f);

    /* Step6 提交绘制 */
    CGContextStrokePath(currentContext);

}
```

如果想连续绘制多条线, 可以再 Code-6中的 Step5和 Step6之间多次调用 CGContextAddLineToPoint().

more:

CGContextSetLineJoin 可以改变线条交叉点的样式.

drawPath

如果我们想快速绘制一条折线, 调用 drawLine 就显得有些臃肿. 所以有了 drawPath, 它是 drawLine 的加强版.

drawPath 的一般步骤如下:

```
    //Code-7: drawPath
    - (void)drawRect:(CGRect)rect{

        /* Step1 获取当期的画布: Graphic Context */
        CGContextRef currentContext = UIGraphicsGetCurrentContext();

        /* Step2 创建 path */
        CGMutablePathRef path = CGPathCreateMutable();

        /* Step3 移动到起始点 */
        CGPathMoveToPoint(path,NULL, screenBounds.size.width,
screenBounds.origin.y);

        /* Step4 绘制一个椭圆 */
        CGPathAddEllipseInRect(path, &CGAffineTransformIdentity,
CGRectMake(0, 320, 320, 160));

        /* Step5 再添加一条直线 */
        CGPathAddLineToPoint(path,NULL, screenBounds.origin.x,
screenBounds.size.height);

        /* Step6 向画布添加 path */
        CGContextAddPath(currentcontext, path);

        /* Step7 设置绘制类型: kCGPathStroke(绘制边缘), kCGPathFill(填
充 path 内区域), kCGPathFillStroke(包含前面两项)*/
        CGContextDrawPath(currentcontext, kCGPathFillStroke);

        /* Step8 提交绘制 */
```

```
        CGContextStrokePath(currentContext);


        /* Step9 release path */
        CGPathRelease(path);


    }
```

more:

常见几何图形的绘制接口:

`CGPathAddCurveToPoint`,`CGPathAddArcToPoint`,`CGPathAddRect` 等等...

transform

iOS 中的 `transform` 使用 `CGAffineTransform` 表示的. 你可以用矩阵方式构造任意二维变换:

```
    /*
    a: The value at position [1,1] in the matrix.
    b: The value at position [1,2] in the matrix.
    c: The value at position [2,1] in the matrix.
    d: The value at position [2,2] in the matrix.
    tx: The value at position [3,1] in the matrix.
    ty: The value at position [3,2] in the matrix.
    */
    CGAffineTransformMake(CGFloat a, CGFloat b, CGFloat c, CGFloat d,
CGFloat tx, CGFloat ty)
```

矩阵表示为:

iOS 也提供了常见变换的快速构建方式:

```
    CGAffineTransformIdentity           //单位矩阵, 不做任何变换
    CGAffineTransformMakeRotation       //旋转
    CGAffineTransformMakeScale          //缩放
    CGAffineTransformMakeTranslation    //平移
```

在 `Code-7 Step4`中, 绘制 `path` 时用到了单位矩阵

CGAffineTransformIdentity，表示不做任何变换．通常情况下，都是在绘制阶段把 transform 作为参数传入．上面提到的 CGPathAddCurveToPoint,CGPathAddArcToPoint,CGPathAddRect 函数都有一个 transform 参数.

iOS 绘图在项目中的应用

通常，我们只需要随心所欲的对 UIView 增加 subView，UIKit 会自动帮我们绘制．但是下列情况下可能需要手动绘制：

1 优化 UITableViewCell 的时候．如果我们的 cell 很复杂，有很多 subView，就会变得很卡顿．就需要手动绘制了．可参考博客：优化 UITableView 性能或者 IOS 详解 TableView——性能优化及手工绘制 UITableViewCell

2 暂未想到，以后想到再说.

原文 http://www.cnblogs.com/jhzhu/p/3498979.html?utm_source=tuicool

ios 开发小技巧之提示音播放与震动

在 ios 开发中，有时候我们需要频繁播放某种提示声音，比如微博刷新提示音、QQ 消息提示音等，对于这些短小且需要频繁播放的音频，最好将其加入到系统声音（system sound）里。



注意：

- 需要播放的音频文件不能超过30秒
- 必须是 IMA/ADPCM 格式[in linear PCM or IMA4(IMA/ADPCM) format]
- 必须是.caf .aif .wav 文件

1、自定义系统声音

```
// 要播放的音频文件地址
NSString *urlPath = [[NSBundle mainBundle]
pathForResource:@"end_refreshing" ofType:@"wav"];
NSURL *url = [NSURL fileURLWithPath:urlPath];
// 声明需要播放的音频文件 ID[unsigned long]
SystemSoundID ID;
// 创建系统声音，同时返回一个 ID
AudioServicesCreateSystemSoundID((__bridge CFURLRef)(url),
&ID);
```

2、根据自定义系统声音时传入的 ID，播放自定义系统声音

```
// 根据 ID 播放自定义系统声音
AudioServicesPlaySystemSound(ID);
```

3、有时候我们需要在完成播放的时候做一些事情，比如继续播放下一个音频或者提示用户，或者决定以后都不再使用该音频 ID 了，可以在下面的函数中

移除该音频 ID

```
        // 播放完成后执行的操作
        /*      *参数说明:
         * 1、刚刚播放完成自定义系统声音的 ID
         *2、回调函数（playFinished）执行的 run Loop，NULL 表示 main run loop
         * 3、回调函数执行所在 run loop 的模式，NULL 表示默认的 run loop mode
         * 4、需要回调的函数
         * 5、传入的参数， 此参数会被传入回调函数里
         */    AudioServicesAddSystemSoundCompletion(ID, NULL, NULL, &playFinished, (__bridge void *)(self));
    }
    @implementation 与@interface 外部自定义一个函数,然后传入这个函数名的地址
    void playFinished(SystemSoundID  ssID, void* clientData)
    {
        unsigned long ID = ssID; // ssID 不能直接作为参数打印出来，需要中转一次
        NSLog(@"播放完成-传入 ID 为-%lu,传入的参数为%@", ID, clientData);
        // 移除完成后执行的函数
        AudioServicesRemoveSystemSoundCompletion(ID);
        // 根据 ID 释放自定义系统声音
        AudioServicesDisposeSystemSoundID(ID);
    }
```



这样就可以实现播放完成后调用预先定义好的函数 playfinished(),继续做一些你希望的事情，但是别忘了，ARC 中 C 语言函数需要自己管理内存，所以这里，我们需要回调完成后移除刚刚添加的操作 ，并且确认当你以后不再需要播放该 ID 的音频时，清理该 ID 音频对应的所有资源

4、完成以上步骤就可以进行简单的通过自定义系统声音来播放简单且频繁的音频文件了。

注意：

1 > 必须等到播放完成才能清理该 ID 对应的所有资源，也就是说，如果我们打算不再使用该音频，可以在完成后调用的函数（playFinished）里清理资源，如上例所示。 不过一般情况下，既然加到系统声音里的多是频繁使用的，可以在程序结束或者某个控制器销毁的时候再清理。

2 > 如果在 AudioServicesPlaySystemSound(ID)之后马上调用 AudioServicesDisposeSystemSoundID(ID)，你将听不到任何声音，并且也不会调用播放完成后的函数， 这是因为，系统音频播放的所有操作都是放到主线程之外执行的，当主线程执行到清理的时候，该音频如果试图播放 ID 对应的音频，将无法找到。

5、哦，还有震动~ 一句话

```
// 播放系统预定义的音效，kSystemSoundID_Vibrate 代表震动，如果有其它需求可以点进头文件里去找相应的提示音类型
AudioServicesPlayAlertSound(kSystemSoundID_Vibrate);
```

原文
http://www.cnblogs.com/liufeng24/p/3505811.html?utm_source=tuicool

MySQL 与 PostgreSQL：该选择哪个开源数据库？哪一个更好？

如果打算为项目选择一款免费、开源的数据库，那么你可能会在 MySQL 与 PostgreSQL 之间犹豫不定。MySQL 与 PostgreSQL 都是免费、开源、强大、且功能丰富的数据库。你主要的问题可能是：哪一个才是最好的开源数据库，MySQL 还是 PostgreSQL 呢？该选择哪一个开源数据库呢？

在选择数据库时，你所做的是个长期的决策，因为后面如果再改变决定将是非常困难且代价高昂的。你希望一开始就选择正确。两个流行的开源数据库 MySQL 与 PostgreSQL 常常成为最后要选择的产品。对这两个开源数据库的高层次概览将会有助于你选择最适合自己需要的。

MySQL

MySQL 相对来说比较年轻，首度出现在1994年。它声称自己是最流行的开源数据库。MySQL 就是 LAMP（用于 Web 开发的软件包，包括 Linux、Apache 及 Perl/PHP/Python）中的 M。构建在 LAMP 栈之上的大多数应用都会使用 MySQL，包括那些知名的应用，如 WordPress、Drupal、Zend 及 phpBB 等。



一开始，MySQL 的设计目标是成为一个快速的 Web 服务器后端，使用快速的索引序列访问方法（ISAM），不支持 ACID。经过早期快速的发展之后，MySQL 开始支持更多的存储引擎，并通过 InnoDB 引擎实现了 ACID。MySQL 还支持其他存储引擎，提供了临时表的功能（使用 MEMORY 存储引擎），通过 MyISAM 引擎实现了高速读的数据库，此外还有其他的核心存储引擎与第三方引擎。

MySQL 的文档非常丰富，有很多质量不错的免费参考手册、图书与在线文档，

还有来自于 Oracle 和第三方厂商的培训与支持。

MySQL 近几年经历了所有权的变更和一些颇具戏剧性的事件。它最初是由 MySQL AB 开发的，然后在2008年以10亿美金的价格卖给了 Sun 公司，Sun 公司又在2010年被 Oracle 收购。Oracle 支持 MySQL 的多个版本：Standard、Enterprise、Classic、Cluster、Embedded 与 Community。其中有一些是免费下载的，另外一些则是收费的。其核心代码基于 GPL 许可，对于那些不想使用 GPL 许可的开发者与厂商来说还有商业许可可供使用。

现在，基于最初的 MySQL 代码还有更多的数据库可供选择，因为几个核心的 MySQL 开发者已经发布了 MySQL 分支。最初的 MySQL 创建者之一 Michael "Monty" Widenius 貌似后悔将 MySQL 卖给了 Sun 公司，于是又开发了他自己的 MySQL 分支 MariaDB，它是免费的，基于 GPL 许可。知名的 MySQL 开发者 Brian Aker 所创建的分支 Drizzle 对其进行了大量的改写，特别针对多 CPU、云、网络应用与高并发进行了优化。

PostgreSQL

PostgreSQL 标榜自己是世界上最先进的开源数据库。PostgreSQL 的一些粉丝说它能与 Oracle 相媲美，而且没有那么昂贵的价格和傲慢的客服。它拥有很长的历史，最初是1985年在加利福尼亚大学伯克利分校开发的，作为 Ingres 数据库的后继。



PostgreSQL 是完全由社区驱动的开源项目，由全世界超过1000名贡献者所维护。它提供了单个完整功能的版本，而不像 MySQL 那样提供了多个不同的社区版、商业版与企业版。PostgreSQL 基于自由的 BSD/MIT 许可，组织可以使用、复制、修改和重新分发代码，只需要提供一个版权声明即可。

可靠性是 PostgreSQL 的最高优先级。它以坚如磐石的品质和良好的工程化而闻名，支持高事务、任务关键型应用。PostgreSQL 的文档非常精良，提供了大量免费的在线手册，还针对旧版本提供了归档的参考手册。PostgreSQL 的社区支持是非常棒的，还有来自于独立厂商的商业支持。

数据一致性与完整性也是 PostgreSQL 的高优先级特性。PostgreSQL 是完全支持 ACID 特性的，它对于数据库访问提供了强大的安全性保证，充分利用了企业安全工具，如 Kerberos 与 OpenSSL 等。你可以定义自己的检查，根据自己的业务规则确保数据质量。在众多的管理特性中，point-in-time recovery（PITR）是非常棒的特性，这是个灵活的高可用特性，提供了诸如针对失败恢复创建热备份以及快照与恢复的能力。但这并不是 PostgreSQL 的全部，项目还提供了几个方法来管理 PostgreSQL 以实现高可用、负载均衡与复制等，这样你就可以使用适合自己特定需求的功能了。

平台

MySQL 与 PostgreSQL 都出现在一些高流量的 Web 站点上：

- MySQL：Slashdot、Twitter、Facebook 与 Wikipedia
- PostgreSQL：Yahoo 使用了一个修改的 PostgreSQL 数据库来处理每天数以亿计的事件，还有 Reddit 和 Disqus

MySQL 与 PostgreSQL 都能运行在多个操作系统上，如 Linux、Unix、Mac OS X 与 Windows。他们都是开源、免费的，因此测试他们时的唯一代价就是你的时间与硬件。他们都很灵活且具有可伸缩性，可用在小型系统和大型分布式系统上。MySQL 在一个领域上要比 PostgreSQL 更进一步，那就是它的触角延伸到了嵌入式领域，这是通过 libmysqld 实现的。PostgreSQL 不支持嵌入式应用，依然坚守在传统的客户端/服务器架构上。

MySQL 通常被认为是针对网站与应用的快速数据库后端，能够进行快速的读取和大量的查询操作，不过在复杂特性与数据完整性检查方面不太尽如人意。PostgreSQL 是针对事务型企业应用的严肃、功能完善的数据库，支持强 ACID 特性和很多数据完整性检查。他们二者都在某些任务上具有很快的速度，MySQL 不同存储引擎的行为有较大差别。MyISAM 引擎是最快的，因为它只执行很少的数据完整性检查，适合于后端读操作较多的站点，不过对于包含敏感数据的读/写

数据库来说就是个灾难了，因为 MyISAM 表最终可能会损坏。MySQL 提供了修复 MySQL 表的工具，不过对于敏感数据来说，支持 ACID 特性的 InnoDB 则是个更好的选择。

与之相反，PostgreSQL 则是个只有单一存储引擎的完全集成的数据库。你可以通过调整 postgresql.conf 文件的参数来改进性能，也可以调整查询与事务。PostgreSQL 文档对于性能调优提供了非常详尽的介绍。

MySQL 与 PostgreSQL 都是高可配置的，并且可以针对不同的任务进行相应的优化。他们都支持通过扩展来添加额外的功能。

一个常见的误解就是 MySQL 要比 PostgreSQL 更容易学习。关系数据库系统都是非常复杂的，这两个数据库的学习曲线其实是差不多的。

标准兼容性

PostgreSQL 旨在实现 SQL 兼容性（当前标准是 ANSI-SQL:2008）。MySQL 则兼容大部分 SQL，不过还有自己的扩展，可以支持 NoSQL 特性，这在参考手册中都有介绍。每种方式都有优缺点。兼容标准会让数据库管理员、数据库开发者与应用开发者更舒服一些，因为这意味着他们只需学习一套标准、一套特性和命令即可。这会节省时间，提升效率，也不会被锁定在特定的厂商上。

支持使用非标准的自定义功能的人们认为这样可以快速采用新的特性，而不必等待标准进程完成。ANSI/ISO 标准在不断演化，因此标准兼容性也是个变化的目标：知名的关系型数据库 Microsoft SQL Server、Oracle 与 IBM DB2也只是部分兼容于标准。

结论

虽然有不同的历史、引擎与工具，不过并没有明确的参考能够表明这两个数据库哪一个能够适用于所有情况。很多组织喜欢使用 PostgreSQL，因为它的可靠性好，在保护数据方面很擅长，而且是个社区项目，不会陷入厂商的牢笼之中。MySQL 更加灵活，提供了更多选项来针对不同的任务进行裁剪。很多时候，对于一个组织来说，对某个软件使用的熟练程度要比特性上的原因更重要。

原文链接

http://www.infoq.com/cn/news/2013/12/mysql-vs-postgresql?utm_source=tuicool

理解事务隔离级别

其中说明了数据库事务有四个隔离级别：read uncommit / read commit / repeatable read / serializable 中文的叫法为： **未授权读（读未提交）/ 授权读（读提交）/ 可重复读 / 序列化**

不论英文或中文的叫法都不大好记忆。那有没有直指本质，容易记忆的方法呢？下面介绍一下我的理解：

以两个事务为例来说明问题。**事务隔离就是对这两个事务的读写并发操作做出限制。**

从数学的角度来看这两个事务的读写有四种排列组合方式： 写写、写读、读写、读读。

那四种隔离级别与这四种排列组合方式是否有联系呢？答案是有。

未授权读就是对两个事务写写的并发控制，授权读就是对写读的控制，可重复度就是对读写的控制，序列化就是对读读的控制。

这里有个隐含的前置条件，写的操作权重是大于读的，也就是说可以写基本上暗含了可以读。

下面从两个事务读写排列组合的角度解释一下事务隔离级别解决了的问题：

写写对应数据库数据更新丢失的问题（写的同时在写）

写读对应脏读(写的同时在读，之后写被撤销)

读写对应不可重复读(读了之后又被修改的数据)

读读对应幽灵数据(读了之后被删除的数据)

这样的解释是否更好理解了呢？

原文 http://zhuyufufu.iteye.com/blog/1997959?utm_source=tuicool

数据库优化实践【高级索引、反范式篇】

经过索引优化，重构 TSQL 后你的数据库还存在性能问题吗?完全有可能，这时必须得找另外的方法才行。SQL Server 在索引方面还提供了某些高级特性，可能你还从未使用过，利用高级索引会显著地改善系统性能，本文将从高级索引技术谈起，另外还将介绍反范式化技术。

第六步：应用高级索引

实施计算列并在这些列上创建索引

你可能曾经写过从数据库查询一个结果集的应用程序代码，对结果集中每一行进行计算生成最终显示输出的信息。例如，你可能有一个查询从数据库检索订单信息，在应用程序代码中你可能已经通过对产品和销售量执行算术操作计算出了总的订单价格，但为什么你不在数据库中执行这些操作呢?

请看下面这张图，你可以通过指定一个公式将一个数据库表列作为计算列，你的 TSQL 在查询清单中包括这个计算列，SQL 引擎将会应用这个公式计算出这一列的值，在执行查询时，数据库引擎将会计算订单总价，并为计算列返回结果。

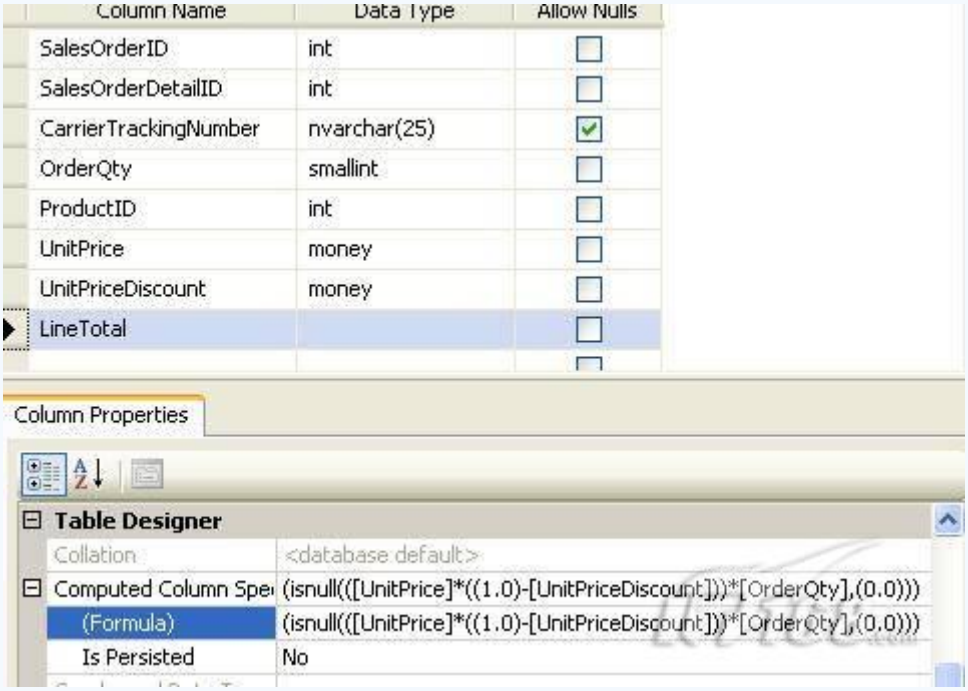


图 1 计算列

使用计算列你可以将计算工作全部交给后端执行，但如果表的行数太多可能计算性能也不高，如果计算列出现在 Select 查询的 where 子句中情况会更糟，在这种情况下，为了匹配 where 子句指定的值，数据库引擎不得不计算表中所有

行中计算列的值，这是一个低效的过程，因为它总是需要全表扫描或全聚集索引扫描。

因此问题就来了，如何提高计算列的性能呢?解决办法是在计算列上创建索引，当计算列上有索引后，SQL Server 会提前计算结果，然后在结果之上构建索引。此外，当对应列(计算列依赖的列)的值更新时，计算列上的索引值也会更新。因此，在执行查询时，数据库引擎不会为结果集中的每一行都执行一次计算公式，相反，通过索引可直接获得计算列预先计算出的值，因此在计算列上创建一个索引将会加快查询速度。

提示：如果你想在计算列上创建索引，必须确保计算列上的公式不能包括任何“非确定的”函数，例如 getdate()就是一个非确定的函数，因为每次调用它，它返回的值都是不一样的。

创建索引视图

你是否知道可以在视图上创建索引?OK，不知道没关系，看了我的介绍你就明白了。

为什么要使用视图?

大家都知道，视图本身不存储任何数据，只是一条编译的 select 语句。数据库会为视图生成一个执行计划，视图是可以重复使用的，因为执行计划也可以重复使用。

视图本身不会带来性能的提升，我曾经以为它会“记住”查询结果，但后来我才知道它除了是一个编译了的查询外，其它什么都不是，视图根本记不住查询结果，我敢打赌好多刚接触 SQL 的人都会有这个错误的想法。

但是现在我要告诉你一个方法让视图记住查询结果，其实非常简单，就是在视图上创建索引就可以了。

如果你在视图上应用了索引，视图就成为索引视图，对于一个索引视图，数据库引擎处理 SQL，并在数据文件中存储结果，和聚集表类似，当基础表中的数据发生变化时，SQL Server 会自动维护索引，因此当你在索引视图上查询时，数据库引擎简单地从索引中查找值，速度当然就很快了，因此在视图上创建索引可以明显加快查询速度。

但请注意，天下没有免费的午餐，创建索引视图可以提升性能，当基础表中

的数据发生变化时，数据库引擎也会更新索引，因此，当视图要处理很多行，且要求和，当数据和基础表不经常发生变化时，就应该考虑创建索引视图。

如何创建索引视图?

1)创建/修改视图时指定 SCHEMABINDING 选项：

```
REATE VIEW dbo.vOrderDetails

    WITH SCHEMABINDING

    AS

    SELECT…
```

2)在视图上创建一个唯一的聚集索引;

3)视需要在视图上创建一个非聚集索引。

不是所有视图上都可以创建索引，在视图上创建索引存在以下限制：

1)创建视图时使用了 SCHEMABINDING 选项，这种情况下，数据库引擎不允许你改变表的基础结构;

2)视图不能包含任何非确定性函数，DISTINCT 子句和子查询;

3)视图中的底层表必须由聚集索引(主键)。

如果你发现你的应用程序中使用的 TSQL 是用视图实现的,但存在性能问题,那此时给视图加上索引可能会带来性能的提升。

为用户定义函数(UDF)创建索引

在用户定义函数上也可以创建索引，但不能直接在它上面创建索引，需要创建一个辅助的计算列，公式就使用用户定义函数，然后在这个计算列字段上创建索引。具体步骤如下：

1)首先创建一个确定性的函数(如果不存在的话)，在函数定义中添加 SCHEMABINDING 选项，如：

```
CREATE FUNCTION [dbo.ufnGetLineTotal]

    (
```

```
    -- Add the parameters for the function here
    @UnitPrice [money],

    @UnitPriceDiscount [money],

    @OrderQty [smallint]

    )

    RETURNS money

    WITH SCHEMABINDING

    AS

    BEGIN

    return (((@UnitPrice*((1.0)-@UnitPriceDiscount))*@OrderQty))

    END
```

2)在目标表上增加一个计算列，使用前面定义的函数作为该列的计算公式，如图2所示。

```
CREATE FUNCTION [dbo.ufnGetLineTotal]

    (

    -- Add the parameters for the function here
    @UnitPrice [money],
```

```
@UnitPriceDiscount [money],

@OrderQty [smallint]

)

RETURNS money

WITH SCHEMABINDING

AS

BEGIN

return (((@UnitPrice*((1.0)-@UnitPriceDiscount))*@OrderQty))

END
```

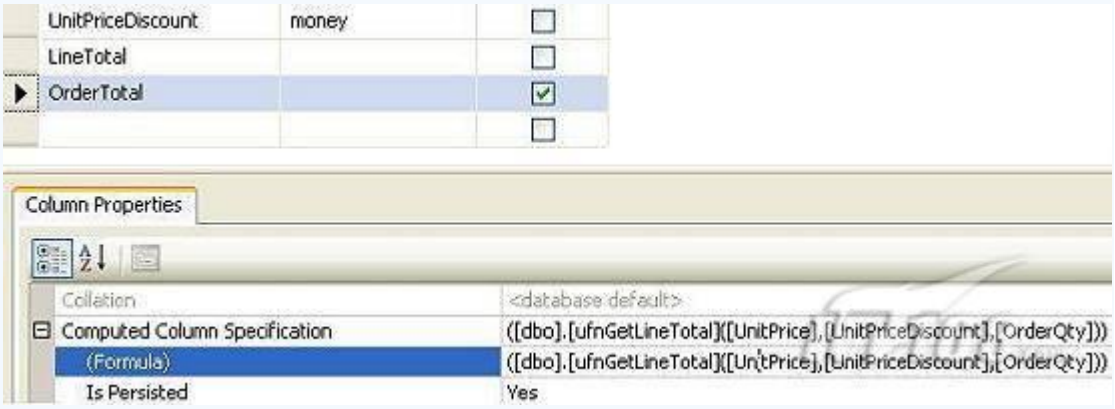


图 2 指定 UDF 为计算列的结算公式

3)在计算列上创建索引

当你的查询中包括 UDF 时，如果在该 UDF 上创建了以计算列为基础的索引，特别是两个表或视图的连接条件中使用了 UDF，性能都会有明显的改善。

在 XML 列上创建索引

在 SQL Server(2005和后续版本)中，XML 列是以二进制大对象(BLOB)形式存储的，可以使用 XQuery 进行查询，但如果没有索引，每次查询 XML 数据类型时都非常耗时，特别是大型 XML 实例，因为 SQL Server 在运行时需要分隔二进制大对象评估查询。为了提升 XML 数据类型上的查询性能，XML 列可以索引，XML 索引分为两类。

主 XML 索引

创建 XML 列上的主索引时，SQL Server 会切碎 XML 内容，创建多个数据行，包括元素，属性名，路径，节点类型和值等，创建主索引让 SQL Server 更轻松地支持 XQuery 请求。下面是创建一个主 XML 索引的示例语法。

```
CREATE PRIMARY XML INDEX
index_nameON <object> ( xml_column )
```

次要 XML 索引

虽然 XML 数据已经被切条，但 SQL Server 仍然要扫描所有切条的数据才能找到想要的结果，为了进一步提升性能，还需要在主 XML 索引之上创建次要 XML 索引。有三种次要 XML 索引。

1)“路径”(Path)次要 XML 索引：使用.exist()方法确定一个特定的路径是否存在时它很有用;

2)“值”(Value)次要 XML 索引：用于执行基于值的查询，但不知道完整的路径或路径包括通配符时;

3)“属性”(Secondary)次要 XML 索引：知道路径时检索属性的值。

下面是一个创建次要 XML 索引的示例：

```
CREATE XML INDEX
index_nameON <object> ( xml_column )
USING XML INDEX primary_xml_index_nameFOR { VALUE | PATH | PROPERTY }
```

请注意，上面讲的原则是基础，如果盲目地在表上创建索引，不一定会提升性能，因为有时在某些表的某些列上创建索引时，可能会致使插入和更新操作变慢,当这个表上有一个低选中性列时更是如此,同样,当表中的记录很少(如<500)时,如果在这样的表上创建索引反倒会使数据检索性能降低,因为对于小表而言,

全表扫描反而会更快，因此在创建索引时应放聪明一点。

第七步：应用反范式化，使用历史表和预计算列

反范式化

如果你正在为一个 OLTA(在线事务分析)系统设计数据库，主要指为只读查询优化过的数据仓库，你可以(和应该)在你的数据库中应用反范式化和索引，也就是说，某些数据可以跨多个表存储，但报告和数据分析查询在这种数据库上可能会更快。

但如果你正在为一个 OLTP(联机事务处理)系统设计数据库，这样的数据库主要执行数据更新操作(包括插入/更新/删除)，我建议你至少实施第一、二、三范式，这样数据冗余可以降到最低，数据存储也可以达到最小化，可管理性也会好一点。

无论我们在 OLTP 系统上是否应用范式，在数据库上总有大量的读操作(即 select 查询)，当应用了所有优化技术后，如果发现数据检索操作仍然效率低下，此时，你可能需要考虑应用反范式设计了，但问题是如何应用反范式化，以及为什么应用反范式化会提升性能?让我们来看一个简单的例子，答案就在例子中。

假设我们有两个表 OrderDetails(ID,ProductID,OrderQty) 和 Products(ID,ProductName)分别存储订单详细信息和产品信息，现在要查询某个客户订购的产品名称和它们的数量，查询 SQL 语句如下：

```
SELECT Products.ProductName,OrderQty

    FROM OrderDetails INNER JOIN Products

    ON OrderDetails.ProductID = Products.ProductID

    WHERE SalesOrderID = 47057
```

如果这两个都是大表，当你应用了所有优化技巧后，查询速度仍然很慢，这时可以考虑以下反范式化设计：

1)在 OrderDetails 表上添加一列 ProductName，并填充好数据;

2)重写上面的 SQL 语句

```
SELECT ProductName,OrderQty

  FROM OrderDetails

  WHERE SalesOrderID = 47057
```

注意在 OrderDetails 表上应用了反范式化后，不再需要连接 Products 表，因此在执行 SQL 时，SQL 引擎不会执行两个表的连接操作，查询速度当然会快一些。

为了提高 select 操作性能，我们不得不做出一些牺牲，需要在两个地方(OrderDetails 和 Products 表)存储相同的数据(ProductName)，当我们插入或更新 Products 表中的 ProductName 字段时，不得不同步更新 OrderDetails 表中的 ProductName 字段，此外，应用这种反范式化设计时会增加存储资源消耗。

因此在实施反范式化设计时，我们必须在数据冗余和查询操作性能之间进行权衡，同时在应用反范式化后，我们不得不重构某些插入和更新操作代码。有一个重要的原则需要遵守，那就是只有当你应用了所有其它优化技术都还不能将性能提升到理想情况时才使用反范式化。同时还需注意不能使用太多的反范式化设计，那样会使原本清晰的表结构设计变得越来模糊。

历史表

如果你的应用程序中有定期运行的数据检索操作(如报表)，如果涉及到大表的检索，可以考虑定期将事务型规范化表中的数据复制到反范式化的单一的历史表中，如利用数据库的 Job 来完成这个任务，并对这个历史表建立合适的索引，那么周期性执行的数据检索操作可以迁移到这个历史表上，对单个历史表的查询性能肯定比连接多个事务表的查询速度要快得多。

例如，假设有一个连锁商店的月度报表需要3个小时才能执行完毕，你被派去优化这个报表，目的只有一个：最小化执行时间。那么你除了应用其它优化技巧外，还可以采取以下手段：

1)使用反范式化结构创建一个历史表，并对销售数据建立合适的索引；

2)在 SQL Server 上创建一个定期执行的操作，每隔24小时运行一次，在半夜往历史表中填充数据；

3)修改报表代码，从历史表获取数据。

创建定期执行的操作

按照下面的步骤在 SQL Server 中创建一个定期执行的操作，定期从事务表中提取数据填充到历史表中。

1)首先确保 SQL Server 代理服务处于运行状态;

2)在 SQL Server 配置管理器中展开 SQL Server 代理节点，在“作业”节点上创建一个新作业，在“常规”标签页中，输入作业名称和描述文字;

3)在“步骤”标签页中，点击“新建”按钮创建一个新的作业步骤，输入名字和 TSQL 代码，最后保存;

4)切换到“调度”标签页，点击“新建”按钮创建一个新调度计划;

5)最后保存调度计划。

在数据插入和更新中提前执行耗时的计算，简化查询

大多数情况下,你会看到你的应用程序是一个接一个地执行数据插入或更新操作，一次只涉及到一条记录，但数据检索操作可能同时涉及到多条记录。

如果你的查询中包括一个复杂的计算操作,毫无疑问这将导致整体的查询性能下降，你可以考虑下面的解决办法：

1)在表中创建额外的一列，包含计算的值;

2)为插入和更新事件创建一个触发器，使用相同的计算逻辑计算值，计算完成后更新到新建的列;

3)使用新创建的列替换查询中的计算逻辑。

实施完上述步骤后，插入和更新操作可能会更慢一点，因为每次插入和更新时触发器都会执行一下，但数据检索操作会比之前快得多，因为执行查询时，数据库引擎不会执行计算操作了。

小结

至此，我们已经应用了索引，重构 TSQL，应用高级索引，反范式化，以及历史表加速数据检索速度，但性能优化是一个永无终点的过程，最下一篇文章中我们将会介绍如何诊断数据库性能问题。

原文 http://www.cnblogs.com/hnlshzx/p/3501302.html?utm_source=tuicool

13.2 优化消息生产者

在代理分发消息到消息消费者之前,消息生产者发送消息到代理的效率是影响整个应用程序性能的基本元素.下面我们将关注几个消息生产者发送消息给代理时,影响消息吞吐量和消息发送延迟的调优参数.

13.2.1 Asynchronous send

13.2.1 异步发送

We’ve already covered the performance gains that can be achieved if you use

nonpersistent delivery for messages. In ActiveMQ nonpersistent delivery is

reliable, in that delivery of messages will survive network outages and system

crashes (as long as the producer is active—it holds messages for redelivery in

its failover transport cache). But you can also get the same performance gain

for persistent messages by setting the useAsyncSend property on the message

producer’s connection factory, as shown next.

我们已经了解了使用非持久化消息分发可以带来性能提升.ActiveMQ 中非持久化消息分发是可靠的,

因为消息会在网络故障和系统崩溃(只要消息生产者依然是活动的--此时消息生产者将消息缓存在失效

转移连接器的缓存中)中幸存下来.但是,通过设置消息生产者的连接工厂的 useAsyncSend 属性,你仍然

可以在使用持久化消息时获得性能提升,请看下面示例代码:

Listing 13.8 Enabling asynchronous sends

代码清单13.8 启用异步发送

```
ActiveMQConnectionFactory cf = new ActiveMQConnectionFactory();
```

cf.setUseAsyncSend(true);

This will set a property that tells the MessageProducer not to expect a receipt

for messages it sends to the ActiveMQ broker. This means that a producer will

not wait until the message is on disk before sending another message.

上面代码将设置一个属性,告知消息生产者不要尝试从代理获取一个刚刚发送的消息的回执.

这就是说消息生产者在等待消息持久化到磁盘之后就不再等待(译注:消息回执)了而是继续

发送下一条消息.

If your application requires guaranteed delivery, it's recommend that you use the

default delivery mode, persistent delivery, and preferably use transactions too.

如果你的程序要求消息精确分发,则推荐你使用系统默认的持久化消息分发模式,并且最好同时使用事务.

The reasons for using asynchronous message delivery for gaining performance

should be well understood, and setting a property on the ActiveMQ connection

factory is a straightforward way of achieving that. Next we'll cover a commonly

misunderstood feature in ActiveMQ: producer flow control. We see a lot of

questions about producers slowing down or pausing, and understanding flow

control will allow you to mitigate this situation in your applications.

使用异步方式发送消息获以取性能提升的原因不难理解,同时以这种方式实现性能提升的方式也

很简单--只要设置 ActiveMQ 连接工厂的一个属性即可.下一节将介绍 ActiveMQ 中一种通常不被

理解的特性:消息生产者流控制.我们将看到大量的关于消息生产者效率下降或暂停问题,并且理解

如何在你的应用程序中使用流控制来减少这些问题的发生.

13.2.2 Producer flow control

13.2.2 生产者流控制

Producer flow control allows the message broker to slow the rate of messages

that are passed through it when resources are running low. This typically

happens when consumers are slower than the producers, and messages are using

memory in the broker awaiting dispatch.

生产者流控制允许消息代理在系统资源紧张时降低消息的通过量.这种情况在消息消费者处理速度慢于生产者时发生,此时消息会缓存在代理的内存中等待被发送.

A producer will wait until it receives a notification from the broker that it has space

for more messages, as shown in figure 13.3. Producer flow control is necessary to prevent

a broker’s usage limits for memory and temporary store space from being overrun,

especially for wide area networks.

消息生产者在收到代理通知有足够存储空间接收更多消息之前都会处于等待状态,如图13.3所示.

生产者流控制对于阻止代理的内存和临时存储空间超过限制的值来说是必要的,尤其是对于广域网来说.

Producer flow control is enabled by default for persistent messages but must be

explicitly enabled for asynchronous publishing (persistent messages, or for connections

configured to always send asynchronously). You can enable flow control for asynchronous

publishing by setting the producerWindowSize property on the connection

factory.

对于持久化消息来说,生产者流控制默认时就是开启的,但是对于异步消息发布来说必须明确的指定为开启(针对持久化消息,或对于配置成总是异步发送消息的连接来说).你可以通过设置连接工厂的 producerWindowSize 属性来开启消息生产者异步发送消息的流控制.

Listing 13.9 Setting the producer window size

代码清单13.9 设置生产者窗口尺寸

```
ActiveMQConnectionFactory cf = new ActiveMQConnectionFactory();
cf.setProducerWindowSize(1024000);
```

The producerWindowSize property is used to specify the number of bytes allowed in

the producer's send buffer before it'll be forced to wait for a receipt from the broker

that it's still within its usage limits. If this isn't enabled for an asynchronous publisher,

the broker will still pause message flow, which defaults to blocking the message producer's

transport. Blocking the transport will block all users of the connection, which

can lead to deadlock if the message consumers are sharing the connection. Producer

flow control allows blocking only the producer rather than the entire

connection.

producerWindowSize 属性用于设置生产者在收到代理的存储空间未超过限制值回执之前可以用来缓存消息的字节大小,超过这个设置的值,生产者将等待代理回执(译注:而不再生产和发送消息).对于一个异步发送消息的生产者来说,如果这个属性未开启,则代理仍然会暂停消息流动,这种情况下,默认会阻塞消息生产的传输连接器.阻塞传输连接器会阻塞所有使用该连接器连接的用户,如果消息消费者也使用同样的连接,则会导致死锁.生产者流控制运行仅阻塞消息生产者而不是整个消息生产者使用的传输连接.

Although protecting the broker from running low on memory is a noble aim, itdoesn't aid our cause for performance when everything slows down to the slowestconsumer! So let's see what happens when you disable producer flow control, as shown in bold in the following code. You can do this in the broker configuration on a destination policy.

尽管保护代理不要运行在可用内存空间低的状态是一个不错的想法,但是这并不能改善系统被最慢的消费者而拖慢时而产生的性能问题.所以,让我们看看禁用生产者流控制时会发生什么,如厦门代码中的粗体字所示.你可以在代理的配置中配置消息目的地策略来实现.

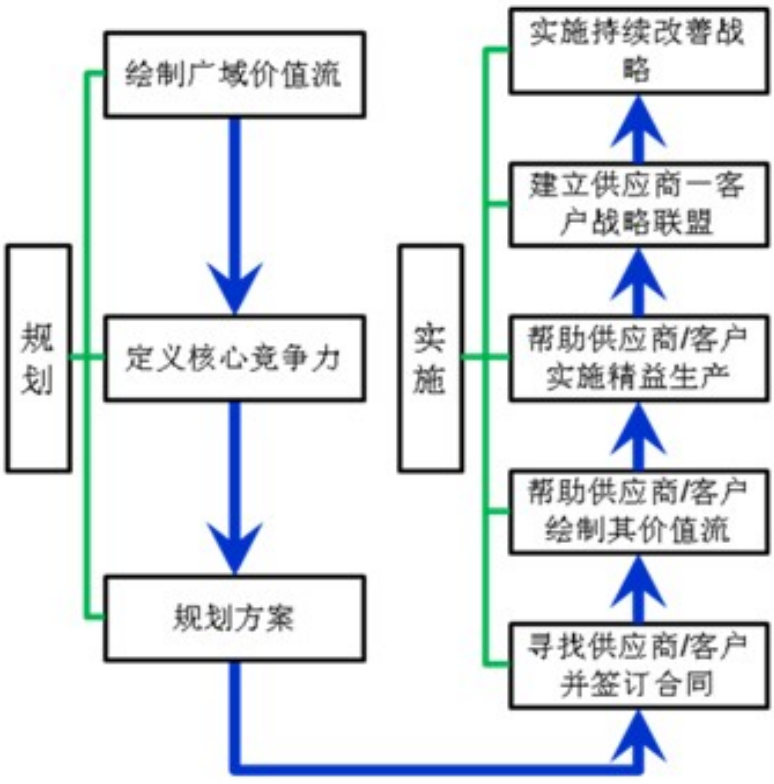
Listing 13.10 How to disable flow control

代码清单13.10 如何禁用流控制

```
<destinationPolicy>
  <policyMap>
    <policyEntries>
      <policyEntry
          topic="FOO.>"
          producerFlowControl="false"
          memoryLimit="10mb" />
    </policyEntries>
  </policyMap>
</destinationPolicy>
```

With producer flow control disabled, messages for slow consumers will be off-lined to temporary storage by default, enabling the producers and the rest of the consumers to run at a much faster rate as outlined in figure 13.4. Additionally, the system usage memory limit determines the point at which messages are offloaded to disk by the pending message cursors. The system usage memory limit setting is applied across the broker. This limit needs to be lower than the destination memory limits so that they can kick in before producer flow control.

默认情况下,禁用了生产者流控制之后,发送给速度慢的消费者的消息会被存放到临时存储控件中,以便消息生产者和其他消息消费者可以尽可能快的运行,如图13.4所示.另外,代理可用的最大内存限制决定了在什么时候消息会通过追加到消息游标的形式持久化到磁盘上.系统可用最大内存限制作用范围是整个代理.这个限制应该要低于消息目的地可用内存限制,以便在流控制之前起作用.(译注:即,这个较小的值先起作用,则消息目的地使用内存不会超过配置的限制值,因为这个值较大)



Disabling producer flow control enables messaging applications to run at a pace independent of the slowest consumer, though there’s a slight

performance hit in offlining messages. In an ideal world, consumers would always be running as fast as the fastest producer, which neatly brings us to the next section on optimizing message consumers.

尽管有一些因为存储消息而代理的性能损失,在禁用了生产者流控制之后,消息应用程序可以独立于最慢的消息消费者而运行.在理想情况下,消息消费者总是与消息生产者一样快速运行,这就给我们引入下一节中关于消息消费者的优化.

TUNING PRODUCER FLOW CONTROL

生产者流控制调优

By default, when producer flow control is enabled and there's not enough space in the broker for more messages, the producer's send operation will block until space becomes available on the broker. There are two ways to tune this parameter so that it doesn't block indefinitely and essentially hang the producer until space becomes available.

默认情况下,当启用了生产者流控制后,当代理没有空间存放更多消息时,生产者发送消息的操作会被阻塞知道代理有足够控件存储消息.有两中方式调整该参数,使得代理获取更多存储消息控件之前,消息生产者不会无限期实质性的挂起.

The first tuning option for producer flow control is named sendFailIfNoSpace:

第一种调节消息生产者流控制的方式称为 sendFailIfNoSpace

```
<systemUsage>
  <systemUsage sendFailIfNoSpace="true">
    <memoryUsage>
      <memoryUsage limit="128 mb"/>
    </memoryUsage>
  </systemUsage>
</systemUsage>
```

The sendFailIfNoSpace property puts control back into the hands of the producer by throwing an exception on the client side when a send

operation is attempted and space isn't available, instead of blocking the send operation indefinitely. This allows the producer to catch the exception, wait a bit, and attempt the send the operation again.

sendFailIfNoSpace 属性将控制权返还给消息生产者,在代理的消息存储已经不足而生产者仍然尝试发送操作时,通过在生产者客户端抛出异常来代替永久性的阻塞生产者的发送操作.这就允许生产者可以捕捉这个异常,然后等待一段时间后,继续尝试发送操作.

The second tuning option for producer flow control was made available in ActiveMQ 5.4.1. This property is named sendFailIfNoSpaceAfterTimeout:

第二个调节生产者流控制的属性开始于 ActiveMQ 的5.4.1版本.该属性名称为

sendFailIfNoSpaceAfterTimeout:

```
<systemUsage>
  <systemUsage sendFailIfNoSpaceAfterTimeout="5000">
    <memoryUsage>
      <memoryUsage limit="128 mb"/>
    </memoryUsage>
  </systemUsage>
</systemUsage>
```

The sendFailIfNoSpaceAfterTimeout property provides a slightly different kind of control. This property causes the send operation to fail with an exception on the client side, but only after waiting the given amount of time for space to become available.

sendFailIfNoSpaceAfterTimeout 与前面那个属性稍有不同. 配置了该属性后,在等待配置的时间后,如果代理端依然没有足够的空间存储消息,则会在客户端客户端发送消息时抛出异常.

原文 http://jackyin5918.iteye.com/blog/1998422?utm_source=tuicool

一个程序员的恐惧

在过去几周，我写了些关于恐惧对程序员（一个人）的影响的内容，这里有一些我所喜欢的引述：

- 当一个团队承认了失败但没有受到责备或遇到尴尬，再次尝试的恐惧会消失，创造力也将激发。
- 在海上，我们害怕火、岩石和溺水，但这些都无法与我第一次坐在办公室上班感到的恐惧相比。
- 我学会去接受那些压抑我创造力的人，以及在世俗上努力取悦他们
- 对于混乱的编码，偏爱优雅代码的我，鼓励自己在周围创造了一个保护罩。
- 我的保护罩由一副耳机和技术语言组成，旨在疏远他们而不是为了交流得更清晰。
- 如果在一个团队讨论里没有感觉到恐惧，我就会以开放的心态倾听。思维的活跃性和每个人的作用将会得到增强。团队的创新力也将得到提高。
- 在这种非正式的社会等级里，我渴望处于一个合理的位置，即能够平等地对待每一个人和敞开心扉地倾听。
- 没有了恐惧，我们能够找到更好的方法去工作。我们每天都做，一做就一整天。我们所做的“工作”正如找到的方法。
- 让我告诉你我对平等的强烈信念，思考我的思想，我将决定是否应该尊重或者轻视你。
- 仔细思考在我一个拥护性团队的工作表现。恐惧和竞争往往导致嘲讽和神气。
- 当我们逐渐成长为一个团队之时，恐惧所引发的不正常行为开始消失，支持和同情随之而来。
- 恐惧可以完全支配我们的行为。它停止我们的思考、帮助、学习、发现和共享；妨碍了我们大量需要的东西。

原文链接

http://www.linuxeden.com/html/news/20140105/147191.html?utm_source=tuicool

如何培养创造力：别学编程，学绘画

本月早些时候，塞缪尔·阿贝斯曼（Samuel Arbesman）在 Wired 网站撰文称，这个世界需要更多的通才、业余爱好者和多面手。他指出（他当然不是第一个指出的人），科学和技术知识的领域变得如此之大，没有一个人能够通晓一切。因此，人们往往会专注于一个领域。他写道，这是个问题，因为“最令人兴奋的发明都出现在各个学科的边缘，出现在那些能够把不同领域的不同观点综合在一起的人当中。”

爱因斯坦拉小提琴。 阿贝斯曼认为，为了培养多面手，更多的人应该“拥抱机器”。尤其是通过学习编程来实现这个目标。阿贝斯曼表示，“通过编程，并认识到相似的算法会不断重复出现，我们能够看到不同知识领域之间的相似性。”

如果你的职业是写代码，那么编程当然是个实用的技能。这也可以是个有益的爱好——过去几年中，我学会了几种不同的程序语言。但是对大多数人来说，学习编程恰恰并不值得花费时间。科技变化如此之快，如果想在编程领域保持领先地位，你就要专注于编程——从而让你自己局限于极为专业的桎梏，这正是阿贝斯曼担心的问题。

（更新：阿贝斯曼非常热情地在本文下方发表了评论，指出他的观点“并不是着重强调编程能够培养多面手这种司空见惯的想法。”而是在说，“目的是让人了解跨越学科的通用模型，而不是学习如何编程。编程也许会有帮助，但是更多的是学会寻找和使用‘工具’，促进这种交互作用的产生，不仅是学习编程本身。”不过每个人都应该编程的这种想法是再普通不过的观点，依然值得探讨。）

在21世纪，要培养在任何领域——无论是科学、技术还是商业领域——的创造力，对科技都应有一定程度的掌握。不过要充分利用计算机的强大功能，最好的途径并不在于编程——而是让计算机做人类并不擅长的繁重计算工作，让人类能够专注于具有创造力、解决问题的工作，这恰好是计算机不擅长的领域。

如果你想培养创造性的以及能解决问题的能力，解决方案并不是学习编程——而是学习绘画雕刻，或者是乐器演奏、诗歌创作。从事什么领域并不重要：关键在于如果你想培养自己的创新能力，最好的方法就是认真地追求一门艺术。

在诺贝尔奖的历史上，几乎每位获奖者都有艺术追求。根据美国心理学家米歇尔（Michele）和罗伯特·鲁特-伯恩斯坦（Robert Root-Bernstein）的研

究，“绝大多数的诺贝尔奖获奖者在成年后都积极参与艺术活动。与普通科学家相比，他们爱好唱歌、跳舞或表演的概率是前者的25倍；同时也身为视觉艺术家的概率是前者的17倍；创作诗歌或文学作品的概率是前者的20倍；做木工活或其他工艺的概率是前者的8倍，同时也是音乐家的概率是前者的4倍；同时也是摄影师的概率是前者的两倍。”

物理学家马克斯·普朗克（Max Planc）写过歌剧，为交响乐谱过曲，他曾经写道，科学家“必须有生动直观的想象力，因为新的创意不是通过推论产生，而是通过艺术创作的想象力产生。”

历史似乎赞同他的说法。实际上，无论是古代还是现代，许多最伟大的科学家都是艺术家。当然，达芬奇（Da Vinci）以他的艺术才华和科学才能而闻名于世。现代蒸汽机的发明者罗伯特·富尔顿（Robert Fulton）是位画家。好莱坞女明星海蒂·拉玛（Hedy Lamarr）与人共同发明的专利技术成为现代手机、wi-fi和全球定位系统（GPS）的基础。谁和她共同完成了这项发明？乔治·安塞尔（George Antheil），一位作曲家和音乐家。

（海蒂·拉玛则从另一个方向体现了这个过程：她的日常工作是名演员，工程师是她的业余爱好。她在自己的豪宅中建立了实验室，在这里完成了多项发明。） 为什么学习艺术可以提高创造力，这个科学原理依然处于起步阶段，有大量的研究产生了相互矛盾的结果。不过有一个原因可能解释这种力量，艺术和科学之间的实践存在相似性。艺术家会先勾勒草图，稍加修改，然后完成绘画，这可能是一幅油画的基础。小说家会创作几篇草稿，然后挑出写得最好的稿子。同样地，科学方法的本质是提出假设、测试、修改假设、形成新的假设、再次测试等等。工程师的工作流程是设计、测试、否决草图、测试、否决更多的草图等等。因此，即使是在某个人选择艺术的实践中，科学家或工程师还是在练习基本技能，这些技能是她在主要职业中获得成功的必要条件。

爱因斯坦曾经说过，“如果我没有成为物理学家，我可能会成为一位音乐家。我经常在音乐中思考问题。我在音乐中做白日梦。我通过音乐的方式来看待我的人生。”为了发挥我们的创造力潜能，我们不妨将爱因斯坦作为典范。

原文 http://www.forbeschina.com/review/201312/0030442.shtml?utm_source=tuicool

Tinyfool 的 2013 年总结————在困惑和挣扎中试图前行

我写过回忆录，虽然没写完。去年元旦过后没有多久，我写过《iApp4Me 一周年记》，在产品和商业上做的一败涂地的前提下，用一些热血的悲情和无奈的慨叹，那篇文章倒也算是赚了不少人的喝彩。

事后，我一直在想，我到底有什么擅长？我写程序真的好么？我写过最长的代码也就是1万来行，还各种乱七八糟的结构。算法不精通，也没有哪一门语言和平台真的是擅长的。

说起来很多人说我是国内很牛的 iOS 开发者，其实无非就算在北京最早玩的100人里面可能有个我而已，真正大卖的 App 没有一个属于我，唯一可以吹嘘的就是帮网易开发过有道词典 iOS 版本的第一个初步的版本而已。知道我们公司的底细的人知道我们 App 里面真正的代码都是 @Scyx 老师和其他的程序员写的，我没有写几句。iOS 刚开始流行的时候，我还录过两集教学视频，现在论坛上问一个基础问题，我可能都会召唤 Sycx 出来帮忙。

公司管得好么？当然管不好。我第一次跟霍矩合伙做搜索的时候，我们就不知道怎么管理和建立销售团队，到了 iApp4me 时期，我也不知道怎么去招聘、管理和做好市场和营销方面的工作，甚至可以坦率的说，我曾经尝试去挖的人，基本上一个都没有挖到过。公司做到人最多的时候，有8个人，其实大多数都是自己送上门来的。到了面试的时候，除了程序员外，其实我也无计可施，怎么考核都不知道。

产品，在自己没有亲自做几个面向最终用户产品之前，还觉得自己是有些独到眼光的。真正做起来，眼光？谁没有，扯皮谁不会？做出来需要多久，有多少前驱后继需要考虑，做出来怎么推广？怎么让用户滚滚而来？我其实还是不会。

文章写的好么？有个一两万 pv 的文章我也写出来过，但是那都是赶上了有什么情绪喷发的时候，我真正谋划要写的东西，往往没有多少人看。从98年到现在，在网上也算是写过不少东西了，有什么真正能流传的么？其实也没有。有没有出过书呢？没有。那一年，参与翻译了两本书，结果是动了手才觉得自己是个废柴，看不懂弄不明白的东西太多，做事情也不够认真仔细，真正到了截稿的时

候，胡乱敷衍了就发了出去。等到参与翻译的书出来，自己的名字不够显眼，甚至觉得有点庆幸，书卖的不好，甚至觉得有点庆幸。就算不提写书，即使是投稿，也只给《程序员》杂志投过几篇很短的东西，这么多年的稿费估计合在一起连5000都没有。

感情问题上基本上也是一个失败者，害怕孤独，但是没有保持长期亲密关系需要的各种特质。谈过屈指可数的几次恋爱，没一个有好下场的。每次甜蜜的时间，都没有后面舔伤口的时间长。日渐苍老以后，有一日对镜自顾，自忖，我要是好人家的女儿，我也看不上这么一个邋里邋遢，身无长物，脾气也不好，没钱没貌的家伙啊。

更严重的是，这一年，开始慢慢感觉到糖尿病日重。有几次，早晨以为老爹做的饭没有盐味，中午外面吃饭发现辣菜没有辣味，咸菜没有咸味，过了一两日才明白是自己莫名其妙味觉消失，养几天吃寡淡的食物才好。又或者，有时发觉自己胳膊上长一些癣，越挠越重，头上长疖子，脸上也莫名的出油。最近一年，牙也开始日益恶化，有时候还感冒一次就一两个星期都好不利索，等等。后来仔细观察联系，其实各种不舒服都跟没有控制好饮食有关系。其实我父母身体也不怎么好，但是今年下来，倒是感觉他们比我身体硬朗多了。我经常想，会不会哪天死在父母之前呢？孩子就苦了，我这么多年也没有什么积蓄，也没有给孩子留写点什么。

其实以上种种想法，就是那篇《iApp4Me 一周年记》之后不久，我的一些所思所感。

公司剩的钱也在逐渐减少。没有用户量也找不到后面的投资支持。13年的前半年，有很多时间花在找下一轮投资，但是业绩确实不好，谁会看的上呢？我开始后悔，刚开始创业的时候，找的投资额太少，不够折腾的；开始想别的出路，等等。

后来，我开始做 iBook Author（iBooks）格式的兼容阅读器，我觉得这东西非常厉害。一方面，是我想做的东西，另外一方面，暂时 iApp4me 想不出来出路的时候，我也不能让自己和团队闲着。做开发的日子其实是相对来说比较快乐的，开始是 @Sycx 一个人写，后来大头和一凡也都加入一起写。管着管着技术团队，我心情越来越好。我看得到我在技术管理的里面的价值。虽然他们三人真的在写，

我参与的代码极其少，而且我参与的代码其实都完成的不怎么样，有些根本就是开了个头，没有结尾；但是我发现，我可以站在他们三个人之外，做一些调配，站在一个局外的角度，去分析，任务如何分配效率最高。以及，如何构建一个逐步完善的目标，从而达到整个项目可以更快的做好技术验证，每一个步骤都可以交付可以测试使用的产品。其实，我做了这么多年技术工作，总是喜欢一个人猛冲，很少在一个团队里面做事情，也没有怎么管理过技术团队。但是这个项目，让我发觉我的各种技术都懂一点，眼界非常宽的优点，在架构层面和项目管理层面是非常有价值的。

这个项目一开始是一个不可能完成的任务，仅仅把 iBooks 格式支持的各种特性列出来，我估计就可以吓到非常多的人。包含，但是不限于，各种形式的图文混排，文字自动绕拍带有不规则形状图像的透明部分，支持各种图像和文字特效，支持各种内建的 Widgets，等等。做之前，我跟@sycx 老师商量的时候，他就跟我说这是一个不可能的任务。我说你估计几个人做多久可以搞定。他说，估计需要3-5人做2-3年。我说，估计我们1-2个人做，可以半年做出来。后来，我们确实是在半年左右做出来的。而且我们大概做了一个多月，就卖出去一个授权，10多万块钱，那时候，虽然大多数功能都没有搞定，但是因为我们的设计一开始就是逐步逼近、持续演进，所以，那个版本交付以后，虽然缺乏一些复杂的功能，但是作为基本使用在限制一些特性的前提下使用，完全没有问题。

虽然投资继续不好谈，也没有人能听懂这个东西到底能做什么。但是我还是充满乐观的心情，因为我们做了一个月而已，就卖出了一套，这很像我们做搜索那年，那年也就是做了一个客户一个月以后，第二个客户就主动上门了。

但，一开始这些都不重要，反正产品还没做好，就继续做产品吧。

等阅读器基本做好了，市场和推广仍旧是个问题。出版社，杂志社聊的很多，都没有什么结果。时机、产品等等方面都有问题。

投资方面更惨淡，能看明白这东西是什么的投资商，我感觉完全没有。

7月份，我们被通知新杂志项目2013中国新媒体创业大赛全国总决赛的时候，其实我已经接触了一圈投资人了，我觉得挺失望的。比赛我估计意思也不会太大，不是很想去，不过既然都进了总决赛了，我又觉得不去也不好，就去了。另外我觉得@sycx 同学跟我创业那么久也够辛苦的，也算是带他去杭州旅游一次（这个

死宅男，来了上海几年，唯一去过的地方就是几个漫展和初音未来的演唱会）。

回来后，虽然拿了个第三名，但是也没有特别亮眼的投资 offer，我开始跟投资人商量万一钱花光以后的后事该怎么做。那时候，虽然知道公司已经到了绝境，但是我还是各种不平，觉得产品没问题，投资界认识不到产品的价值不完全是我的问题。投资人劝我当机立断，先裁员，把成本降下来，再做打算，就算要死也不马上死于钱紧。

那其实才是整个2013年最苦闷的日子。

于理员工都是我招募的，都是在我的管理下行事，公司经营不好不是他们的责任。于情大家2年来天天一起做事，一起吃中饭，饭后还经常一起玩，从感情上这个决定也不好做。其实，我脸皮这些年厚了，面子倒不是问题，但是这个举动是不是在宣布投降呢？我之前放弃搜索生意以后，后悔了很久很久，我不想再次不战而退。

所以，几天后，我开始跟员工谈话，商谈后事，最后，从7人收缩到3人的规模，算了算银行账户余额，本来还有2、3个月就要花完的钱，又可以用到将近1年后了。

然后呢？死是暂时死不了了，然后呢？

那是一段第二苦闷的日子。

早死早超生，如果不裁员，大概13年9月份的样子，我们就会清盘，一定会很痛苦，然后，既然我没有什么余钱，我只能马上找份工作。按照现在移动互联网的热度，收入不见得比在盛大少。我个人的经济状态马上会得到大幅度改善。

我真的无数次这么想过，承认我错了，我白痴，我愚蠢，我不够努力。然后，放弃一切鸿图霸业的梦想，找一家可以养家糊口的公司，好好挣一份高工资。尽量每天早点去，不迟到，每天做完自己的事情就闪，多点时间陪孩子，多点时间出去玩，买个房子，买个车，买点自己喜欢的家具，把自己的屋子好好弄一弄，有点家的感觉。

不赶快挣钱的话，按照我身体恶化的程度，也许我会死在40多岁？还够不够给孩子存点钱呢？

那段时间看了不少美剧，反正心烦意乱就看一集新的，看完了，高兴一会儿，还是心烦意乱，然后，再看再乱，再乱再看，一天就过去了。

后来，这样晨昏颠倒的看美剧，睡觉，吃饭，看美剧，没有多久，我发现身体恶化的太厉害了。我开始考虑该怎么办。

想到的第一个问题是我想活下去，我想看到郝依然长大。第二，我不想混一年，我不知道没有钱了，我们怎么做一番大事业，但是我不想庸庸碌碌待1年。第三，我想出去见见世面。

我开始认真的走路；我跟 @sycx 老师沟通开发进度，开始研究我们当时觉得非常麻烦的把我们的阅读器移植到 Mac/Android 和 Windows 平台的方法；我趁8月份同学聚会的机会，回趟天津的机会，办了护照和港澳通行证，开始策划，香港游、泰国游和美国游。

开发进度是 Mac 版本我们其实已经做出来了，Android 和 Windows 平台下阅读器的开发已经有了很大的突破，而且副产品可能能单独挣钱。

旅游的进度是，我去了一趟香港，还想再去一次，泰国还没成行，美国也没有。

另外一个成效是，我们最近跟一家公司签订合同，把我们的技术用于国内的平台电子教材项目里面，2014年的营收基本上可以持平了，又要多活一年。

还有一个成效是，Tiny4Cocoa 论坛从我们裁员后，我有精力去经营了，现在流量和用户慢慢的持续增长，也开始有一些很忠诚的用户了，我们在上海地区搞的线下小活动也很多次了。

相比几个月前，有好几件事情都有点成效，但是困惑和挣扎依旧。

最近，我开始认真的考虑移民的问题。我的公司如果立足于做技术解决方案的话，我人在上海还是北京，美国还是东京，其实差异不大。所以，我有移民的考虑了。本来是准备真的不行了，清盘了在考虑的。

我不知道明年的今日会不会不再迷惘和彷徨，只希望能多养成好的习惯，以后多自省，不是一年一自省，而是日日自省。我希望还能静下心来，日拱一卒，说起来去年说是这么说，很多时候，还是在迷失在迷惘和彷徨之中，唯一有点收获的就是一点一点做出来的阅读器项目，那确实是日拱一卒，从一个完全没有影子的项目一点一点做出来的。希望到2014年，写总结的时候，能有些辉煌欢乐，而不是悔恨和悲情，与大家共勉。

原文 http://tiny4.org/blog/2014/01/tinyfool-2013/?utm_source=tuicool

技术人攻略访谈十九：iOS 大 V 养成记

对于任何人来说，分享都是一件需要花费大量时间和精力的事情，对唐巧也不例外。白天要面对创业公司高强度的工作压力，晚上和周末才能挤出时间。在难以看到短期成效的情况下，投入精力去做一件付出巨大，“利他”大于“利己”的事情，无疑是一场考验。这个过程中的咬牙坚持，辗转反复，和运动极限遇到的挑战类似。但那些最终战胜自己，顺利通过这场考验的人，将有机会见到更多的风光，遇到更好的自己。

- **技术人攻略：你从什么时候开始做 iOS 开发？为什么会进入这个领域？**

2010年我研究生毕业加入有道，先后做了网易邮箱底层的 API 开发，和网易微博上的实时搜索系统。由于微博没做起来，加上我看好移动互联网的发展，就申请转到有道云笔记组做 iOS 开发。我之前对 Java 很熟，但没有 iOS 开发的背景。有道的技术氛围很好，允许员工转向新的领域。公司的技术评级制度就鼓励员工精通两门以上的语言，达不到就无法涨级别。

当时是11年5月左右，公司给了我一个月时间学习 iOS 开发。于是那一个月几乎没干活，天天看斯坦福的视频和代码，一个月之后就能上手了。接下来用了三个月时间，从无到有开发了有道云笔记的 iPad 版。做完 iPad 版，又给 iPhone 版增加了手写、涂鸦、密码保护等新功能。

- **技术人攻略：为什么选择离开有道去创业？**

刚加入有道的时候，很喜欢它的工作环境，牛人很多，很适合技术人成长。但待了一年之后，发现公司的技术氛围开始变化。有道的领导基本都是技术精英，缺乏懂营销、推广和管理的人才，很多产品做的很好，但是都没成，加上搜索没做起来，离职同事越来越多。资金投入的减少导致新人的质量下降，原来良好的技术交流氛围也消失了。部分新人对技术没有热情、公司内 iOS 的技术交流搞不起来、技术上没有机会做更深入的东西，这几个原因直接导致了我想离开有道。

本来计划等合同期满了之后再换公司，而且我有网易的股票，9月份可以兑现。但去年5月份的时候，遇到了现在这个机会。我当时想的是，即使创业不成功，我也会经历一个 iOS 产品从无到有的过程，这对我来说是一个很好的机会。并且团队的同事都曾经合作过，大家都特别喜欢技术，和一群兴趣相投的人一起工作会很开心。于是放弃了手上的股票，加入这家创业公司。

- **技术人攻略:你经营的关于 iOS 开发的微信公众账号听说已经超过了5000粉丝，当初为什么想要做这样一个公众账号？是怎么做到的？**

微信公众账号的诞生和前期的积累息息相关，这还得先从写博客说起。在做 iOS 开发之前，我就已经开始写技术博客，当时还没有开个人博客。iOS 开发是个新生事物，三年前会的人很少。我发现没人谈这个，而这又是个快速发展的领域。写博客一来可以树立个人品牌，提高自身的影响力，二来能给整个社区贡献内容。Java 开发的任何内容都已经有人写过了，网上一搜一大堆。而因为没人谈 iOS 开发，原创的博客资料就越发珍贵。写了几篇之后，就干脆独立开了一个博客。

这样写了一年博客。加入创业公司之后，由于做 iOS 开发的只有我一个，所以当时很恐惧，害怕失去和大家交流的机会，于是建了一个 QQ 群。首先把我以前的同事都加进来，然后通过同事的人脉圈子，想办法把其他公司做 iOS 开发的人都加进来，拉拢了大概20个人。这20个种子用户中，其中有几个的水平特别高，也很喜欢分享和讨论，所以每天都会有一些讨论出来。我感觉讨论的内容实在太好了，如果不分享给大家有点说不过去。正好去年春节的时候微信推出了公共帐号功能，就建了一个。

于是每天下班之后，我会把群里讨论的内容做一些梳理，通过公众账号发出去。不夸张的说，我是第一个在公众账号中起 iOS 开发这个名字的，所以那段时间涨粉涨得特别快，很快关注的人数到了1000人。公众账号运营三个月之后，冯大辉和池建强分别推荐了一下，又迅速增加了1000粉丝。

这些优秀的内容吸引了很多牛人申请加入 QQ 群，当群人数加到30人的时候，我开始有意识地做一些控制。例如让申请人提供真实姓名、APP 作品、Github 帐号、博客地址，我会看他的博客和 Github，并且找认识的人核实他的基本情况。就这样 QQ 群慢慢加到了50人，成员覆盖了从大型互联网公司、中型互联网公司、创业公司、自由职业及海外 iOS 开发人群。**这个群已经产生出一种生态效果，对于任何的行业问题，都已经足够自行讨论出一个明确的、有结果的、正确的答案了。**

微信公众账号的粉丝也超过了5000人。估计国内有10万的 iOS 开发者，当然活跃的人可能到不了这么多。我希望能将公众账号粉丝发展到2万，覆盖差不多

20%-30%的 iOS 开发人群，按现在的增长速度还需要一两年才行。

- **技术人攻略：能有这么多粉丝和你坚持写了两年技术博客关系很大。你写博客的动力是什么？每次更新需要花多长时间？**

坚持写博客可能和我原来的经历有关系，我曾利用暑假去福建支过教，教小孩计算机。**我觉得把知识传达给别人是一个很舒服的过程，某种意义上来说，你的思维延续到他们身上，感觉像是你的生命延续到他们身上了。**支教的时候这种感觉特别强烈，我对计算机的热爱传递到学生身上，是一种特别有成就感的事情。

写博客还有一个原因，感觉技术男很容易不善沟通，我想通过写博客，锻炼自己沟通、语言组织方面的能力，这也是自我完善的一个过程。我平均一个月发两篇博客，每写一篇要四个小时。一般是周末下午开始写，写到吃晚饭。吃完晚饭之后，修改一下发布出去。

最近刚和出版社签约，打算写一本 iOS 开发的书。国内 iOS 提高类的书比较少，翻译的质量也不高。这本书预计要写一年，明年年底交稿。虽然投入跟产出根本不成正比，但还是愿意凭兴趣做事。

整个中文世界的技术资料跟英语世界是没法比的，基本上现在 iOS 开发的资料还得通过 Google 查找。国内真正愿意写文章的人太少了，我觉得有两方面原因，第一是有的从业者对技术并不是真正感兴趣，另外是中国的教育过程不强调个人的写作和表达，很多牛人做得很好，也很喜欢自己做的事，但不喜欢写，不喜欢分享。这就造成国内的技术资料非常少，交流圈子小。

- **技术人攻略：程序员应该提升自身在技术圈的影响力吗？个人的影响力究竟有多重要？**

非常重要，但是很多人没有意识到。阿里非常强调技术人员在社区的影响力，在评级的时候会把这个因素考虑进去，这是第一个把影响力作为评级标准的大公司。**当你有了一定的影响力之后，有可能推动一项技术的发展，或者改变大家对某个技术观念的看法，不仅可以推动公司的项目，甚至还有可能改变整个技术生态的走向。**

程序员个人塑造自我影响力能得到很多好处，首先是可以提升自我的表达能力，这对自我是很好的成长；其次是可以结识很多同行，而且是兴趣相通的同行，

你的人脉关系和交流圈子将得到扩大；第三是当你有想法推动一些事情的时候，影响力或许可以帮助你完成这件事。

我私下里问过几个技术水平很高的同事，为什么不写点博客、去参加一些技术分享。他们感觉表达是一件很困难的事情。其实我之前的表达能力也比较差，高考语文150分才考了90，但是我一直坚持写东西。当然我更多是把分享技术当兴趣来做，因为从开始分享到得到好处，这个过程见效非常慢，如果真是要功利地在利益驱动下做，还真挺难坚持的。

- **技术人攻略：作为 iOS 开发高手，能不能介绍一下这门技术有什么样的特点？菜鸟要如何才能成长为开发高手呢？**

iOS 开发属于入门比较快，但是精通比较慢的技术。一个月入门，三个月熟练，但有很多深入的技能需要一段时间的积累，这个积累我个人认为需要三到五年，我自己现在也没有达到。

客户端开发最重要的是保证流畅的交互和合理的 UI 设计，用户用起来不会卡，并且感觉很舒服。程序员需要跨界地了解一些 UI 用户体验方面的东西，并在有限的内存和 CPU 的限制下，把这些效果实现。同样的效果在 iOS 设备上实现的很好，在 Android 上可能就很难做。每个设备都有自己的特点，开发受限于操作系统，所以过往的经验很难去复制，需要掌握特有领域的一些知识。

新入行的人如果有钻研精神的话，可以通过工作中遇到的问题慢慢积累。**思维习惯会决定这个人能走多远，例如网上搜到一段解决问题的代码，有些人会去研究，为什么这段代码能工作，背后的原理是什么，不满足于简单地解决问题，而是通过深入思考来积累知识。这可能暂时对工作没有直接的帮助，但是长远来说这两种思维方式会造成巨大的差距。**为什么大部分人总是提出那些很初级的问题，因为他从来不去思考。

另外，我还是个传统的博客阅读者，会订阅很多国外的博客，通过阅读博客积累一些东西。我觉得这也是一个很好的学习方式。

- **技术人攻略：你未来一到两年的目标是什么？**

我想成为 iOS 方面的专家，刚才说这个过程需要3-5年的积累，我感觉自己掌握得还不够深。通过写书，持续发博客和微信公众帐号，作为我的驱动力，让自己可以持续提高。我其实还想了解 Android 开发，但希望 iOS 开发达到一定高

度的时候再去做。我想研究跨平台的解决方案，移动开发最终很有可能形成一个大统一的开发的模式，通过一套方案来同时解决 iOS 和 Android 开发。要解决跨平台的问题，必须精通代码在各个平台下运行的差异性，能做这件事的人，一定需要同时精通 Android 和 iOS 开发。

- **技术人攻略：现在有这方面的解决方案了吗？iOS 开发还有哪些新趋势？**

跨平台目前还没有很成熟的方案，大家都在尝试。豆瓣刚出的豆瓣音乐人，用了 PhoneGap 框架，可以直接编译成 iPhone、Android 和 Windows Phone 的代码，能极大地提升开发效率。HTML5有这么高的关注度，就是因为它能解决跨平台开发的问题，现在 HTML5在手机上的体验还不是很好，苹果比较强调用户体验，它更倾向于你用原生的代码来实现，实现交互的效果会好很多。

还有一个趋势就是远程动态更新客户端的代码。APP Store 的审核期限从一天到十天不等，增加一个功能，或修复一个 Bug，需要提前一两周。通过新的技术手段，让 APP 具备自我升级功能，能够自动去更新内部的逻辑代码。这个技术目前也有一些不太成熟的实现方案，PhoneGap 框架在一定程度上可以做到，通过 HTML5重新下载一个新的网页，就可以改编之前的内容。还有通过 Lua 实现的，或通过一个类似于 HTML5和 CSS 配置来动态改变 APP 的内容。目前这些技术方案都还不是很成熟，没法做到通用性，也没法保证很好的性能、开发质量和速度。

- **技术人攻略：谈谈你现在的情况吧，创业一年多了，这个过程中有什么收获？**

这次创业我不是合伙人，我的目标更多是通过这个创业项目来锻炼自己。通过参与一家公司的创建和发展，在这个过程中学习公司如何运转、如何做决策、会经历哪些困难，为下一份真正的创业积攒经验。加入这家公司之前，我很担心创业公司的管理是混乱的，没有章法，什么都做，什么都不精，天天累得要死。加入之后才发现，**其实很多创业公司是小而美的，同一段时间在创业公司收获的成长和回报，会远远高于大公司。如果不是出来创业，我不可能在技术上获得那么多收获成长，也很可能不会那么积极地跟他人交流和维护公众账号。选择离开网易，虽然损失了一些工资和股票，但是和回报比起来不值得一提。**我挺鼓励有志向的人创业，特别是趁着年轻的时候。

从最开始的教育社区粉笔网，到现在的猿题库，公司在发展过程中也遇到一些坎坷，但我们及时调整了产品方向。整个技术团队也是小而美的，保持着非常高效的工作效率，我们合作得非常开心。公司的社保和公积金都按最高比例缴纳，去年末还发 iPad Mini，既有大公司的福利，但同时又保持小公司的创业气氛。

- **技术人攻略：你在大公司和创业团队都工作过，它们最大的区别在哪？**

最大的区别是创业公司的工作更加高效。首先事情变得更多，如果想要不加班，就需要想办法尽量高效地去做事情。我们会用敏捷 Scrum 的方式来安排未来的工作计划，做粉笔网的时候，完全按照预期的三个月交付。我觉得敏捷方法要成功实施，关键还是看团队文化和团队成员的向心力。有人说过，只要团队牛，不管用什么管理方法都能成功，因为团队成员会自发做一些高效的沟通，管理只是辅助手段。

举例来说，为了防止对他人工作的打断，我们还采用了 Wiki 来记录重要的事情，基本上默认任何事都能在 Wiki 上找到解答。比如说服务端做了一个接口，肯定会写在 Wiki 上，等开发完了需要调试时候，其它同事可以去 Wiki 上看这个接口，按接口去调试。对程序员来说，一次打断就是很大的时间消耗，要接上上一次的思路至少要十分钟。如果可以拥有整段不受打断的工作时间，效率会有极大的提高。

大公司开会太多，同事间沟通常用 QQ，工作状态经常会被是分成小段时间，常常是下班了才有时间写代码。大公司的大部分时间都是消耗在沟通上了，你会觉得生命被浪费。但在小公司，工作效率的提高会带来个人很大的成就感，你会发现一天干的活可能比大公司三天或五天还多，你的成长也会非常快，就会觉得很舒服、很满足。

大公司有太多的体制和制度来局限一个人的发展，KPI 制度、晋级制度、沟通成本等，很多人觉得这就是一份工作，或者即使有热情投入进去，努力也很难得到反馈。创业公司的利益驱动更强，你的付出直接跟回报关联，并且因为生存压力，会更注重减少沟通成本，提升工作效率，保证开发速度和质量的平衡，一切目的都是快速把产品完成。**创业是条不归路，接触过大量以前接触不到的东西，眼界开阔很多之后，就再也难回到大公司做螺丝钉的角色了。**

- **技术人攻略：如果不创业，换一种生活方式，你打算做什么？**

虽然公司业务发展不错，但这次创业远远没到成功那一步。创业公司最怕就是“活死人”，即能挣到钱，但是企业的规模一直无法增长，这就陷入一个两难的境地，是继续投入、继续运营，还是放弃，“活死人”是创业公司最难受的状态。如果这次创业成不了，我可能会尝试另外的生活方式，可能会去做老师，专门教 iOS 开发。现在培训行业很少有像我这样，有大公司工作经验，又有创业经验，而且自己又专心在研究 iOS 技术的人加入。我觉得我当老师应该挺好的，我挺享受教书的状态。如果一个月只收几个学生，带着他们做项目，针对他们的特点教学，他们的收获肯定比在大班里面学到的多。

另一个想尝试的就是做独立的技术咨询，帮公司提供 iOS 解决方案。调研不同公司的需求之后，找出可复用的部分，做成通用的解决方案。第三个想法是做自由职业者，通过自己写 APP 或者接活产生收入，并去不同的城市体验生活。

- **技术人攻略：你平时有什么兴趣爱好？**

我骨子里是比较内向的人，特别宅，就喜欢计算机相关的东西，平时有时间基本上就是研究一下新技术，写写博客。我还对做饭感兴趣，能做一手好菜，一辈子对自己的胃好点，是一件很幸福的事。

- **技术人攻略：这么喜欢计算机，很早就接触编程了吧，做过什么有趣的事情吗？**

我比较喜欢用编程做一些好玩的事情。以前春节回家玩 QQ 上的找茬游戏，玩了一会儿就觉得没意思，于是写了一个自动找茬的程序，把胜率做到了99%，之所以不是100%,是因为我为了把程序做得象真人,把找茬点击的速度降下来了，遇到别人也是机器人的时候，就挂了。

类似的事情还有就是我自己做过 QQ 象棋的外挂程序，也是非常牛逼。基本上输的时候对方一看就是机器人。因为象棋的机器人有一个特点，它的算法是基于搜索和剪枝的，一般情况下，进入比赛盘中的时候，搜索的深度都差不多，于是你就会发现，对方的机器人每次行动的时间都几乎一模一样。看两边机器人下棋还挺有意思的，你说呢？

原文 http://blog.segmentfault.com/devlevelup/1190000000376518?utm_source=tuicool

世界上最美丽的程序员+模特：Lyndsey Scott

Lyndsey Scott 今年才29岁，在模特界却已经有了不俗的成绩。但她还有更多人不为所知的一面——好比超人 Clark Kent 兼备“普通人”和“氪星人”的身份，白天 Scott 为各种知名品牌如 Gucci、普拉达、C&K、维多利亚的秘密等进行模特走秀，而晚上她却变身为可能是世界上最美丽的程序员。

据 Scott 自嘲，以前曾经戴着厚重的框架眼镜，“很多次戴着眼镜就睡着了，眼镜压碎了，就在中间绑上了胶袋。”

和 Scott 进行 Skype 采访时，面对眼前的模特，很难想象她趴在电脑前敲打代码的样子。但这正是这样两种身份的差异之大，让我觉得她充满神秘的魅力：她同时颠覆了典型程序员和模特的印象。

2006年 Scott 从阿默斯特学院毕业，获得计算机科学和戏剧双学位。“计算机的成绩比戏剧要好，我学的很快”，Scott 自述。

Scott 刚开始模特事业时，家人还不大放心，总是介绍计算机方面的工作给她。“大学毕业头两年，家长总是会从门缝里面塞各种招工信息，都是他们了解到的各种计算机科学相关的。”

但她还是顶住了压力，2009年初模特事业开始有所起色，与 C&K 建立合作关系，并成为该公司在纽约时装周全面代言的第一个非裔美国人。

从此 Scott 在模特工作上突飞猛进，先后为普拉达，Gucci，路易斯威登，Vera Wang，Fendi 等知名品牌代言模特。其中最耀眼的成绩大概要数被选上为维多利亚的秘密走秀。

Scott 是家中四个孩子的老大，最小的弟弟 Matthew Scott 说姐姐是家庭的先锋模范人物，“如果要把她比作什么人，大概是 Gisele Bundchen 和比尔盖茨的综合体。说这两个人加合起来大概难以想象，但 Lyndesy 确实就是这样的存在。”

虽然一直以模特为主业，Scott 并没有间断过编程。业内很少有人知道 Scott 自学了 Phython，Objective C，iOS 编程，自己做过7、8个应用，其中2个还能在苹果商店下载。Scott 表示还没有碰到像自己这样能从事编程的模特，但也并不会刻意告诉别人自己这一爱好。

她的第一个应用是为慈善组织 Educate!编写的宣传 APP，发布在 iOS 平台，

Educate!帮助非洲年轻创业家发展。第二个公开的 iOS 应用名叫 iPort，是专门为模特设计的移动电子资料夹。Scott 解释说，自己做这些应用是出于实用的目的，“纸质资料容易遗失，不好整理”。Scott 还将应用推荐给经纪人和其他模特，但目前并不打算把编程发展为主业。最近她在编写第三个应用，并且保持神秘不愿透露相关信息，但提示能帮人发展私人和工作等各种人际关系。

Skype 采访中，Scott 还透露自己并不确定是否应该将“程序员”的第二身份向模特界公布：

“模特界正不断缩减模特的数量、简化团队。包装我的最好方式是突出比我实际年龄更年轻的一面。他们不会谈论包括学历在内的真实的我是怎么样的。这一点我很理解。在模特这一行年轻比大学学历更有价值。”

Scott 是 Stack Overflow、Quora(均为 IT 类技术问答网站)的常客，最近她在 Quora 网站上回复了一则提问，这也是她第一次在 Quora 上回答问题。“当时我正在熬夜编程，Quora 上蹦出了这个问题”。回答中 Scott 附上了一张小时候带着紫边宽边眼镜的照片、一张抱着足球穿着运动服的假小子的照片，以及不久前在维多利亚的秘密走秀台上和后场的照片。



问题：我们都知道丑小鸭到白天鹅的故事，真实社会中你有过这样的体会么？你觉得自己因为长相而受到优待么？

Scott 回答中提到：

“对比之前和现在人们对我的态度，变化大的让人震惊。青少年时期被欺负

是家常便饭，朋友也很少，原因大概是人们戴有色眼镜看人吧……而现在这种态度的变化，可能是因为他们发现我特别的一面。现在他们给我机会，却会时不时失望地发现原来我还是会表现出书呆子的一面(笑)。”

Scott 的回答在 Quora 上获得了三千多个赞，并且被疯狂转载，甚至 Slate 也全篇报道了这一回帖。Scott 坦言现在也还是能会有某种“被欺负”的经历，虽然这次是来自行业内的专家：“有不止一家经纪人告诉我我需要表现的更加职业。当然这些话让我觉得自己应该做的还不够好。”

虽然在 Quora 上 Scott 提到了自己程序员的工作，但只是一笔带过。“毕竟这不是我的主要职业，过于公开谈论不太好。”

Scott 采用非常低调的方式逐步让自己的另一面为人所知。已经有以为来自 Elite Model Management 的经纪人 Tony Vovroch 支持她这么做，并同意她自由谈论自己的第二身份。

Vavroch 认为：“只要能增加模特外在美之外的事情，能充实模特的形象，都是好事。形体美是成为模特的基本，但其个性、其他方面女的特长、能力，都会让广告商、客户等业内人士对模特更感兴趣。”

Scott 对这种看法表示认同：“就好像树立个人品牌，我希望能呈现真实的自己。”

原文 http://www.kuailiyu.com/article/7003.html?utm_source=tuicool